МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

**ИВАНОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ИНСТИТУТ МАТЕМАТИКИ, ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК**

«Рекомендовать к защите»

Заведующий кафедрой информационных

технологий и прикладной математики,

доцент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Данилова С.В.

протокол заседания кафедры № \_\_\_\_

от «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(БАКАЛАВРСКАЯ РАБОТА)

**Прогноз яркости точки изображения через предыдущие точки**

|  |  |
| --- | --- |
| Направление подготовки: | 02.03.02 Фундаментальная информатика и информационные технологии |
| Направленность образовательной программы: | Фундаментальная информатика и информационные технологии |
| Выпускную квалификационную работу выполнил: | студент 4 курса очной формы обучения \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Китаев Максим Михайлович |
| Руководитель выпускной квалификационной работы: | доцент кафедры фундаментальной информатики и информационных технологий,  канд. физ-мат. наук,. \_\_\_\_\_\_\_\_ Хашин Сергей Иванович |

Иваново, 2024

Министерство образования и науки Российской Федерации

ФГБОУ ВО «Ивановский государственный университет»

Институт математики, информационных технологий и естественных наук

Кафедра информационных технологий

и прикладной математики

**Утверждаю:**

Зав. кафедрой ИТ иПМ

канд.экон.наук, доцент\_\_\_\_\_\_\_\_\_Данилова С.В.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 год

**ЗАДАНИЕ**

по подготовке выпускной квалификационной работы

студента 4 курса бакалавриата

института математики, информационных технологий и естественных наук

Китаев Максим Михайлович

(фамилия, имя и отчество студента)

Тема работы: Прогноз яркости точки изображения через предыдущие точки

(утверждена приказом ИвГУ № 2п/283 от «29» декабря 2023 г.)

Срок сдачи студентом законченной работы: «16» июня 2024 г.

Исходные данные к работе:

Краткое содержание работы:

Дата выдачи задания: «\_3\_\_» \_\_мая\_\_ 2024 г.

Научный руководитель,

канд.физ-мат..наук, доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Хашин С.И

(подпись)

Задание принял к исполнению, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Китаев М.М.

студент (подпись)

Министерство образования и науки Российской Федерации

ФГБОУ ВО «Ивановский государственный университет»

Институт математики, информационных технологий и естественных наук

Кафедра информационных технологий

и прикладной математики

**Утверждаю:**

Зав. кафедрой ИТ и ПМ

канд.экон.наук, доцент\_\_\_\_\_\_\_\_\_Данилова С.В.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024год

**КАЛЕНДАРНЫЙ ГРАФИК**

выполнения выпускной квалификационной работы

(бакалаврской работы)

студента института математики, информационных технологий и естественных наук

Китаева Максим Михайловича

(фамилия, имя и отчество студента)

Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(утверждена приказом ИвГУ № 2п/283 от «29» декабря 2023г.)

|  |  |  |
| --- | --- | --- |
| Характер и объем работ | Плановый срок выполнения | Отметки о выполнении |
| 1.Сбор информации и изучение литературы | 15.05.2024 |  |
| 2.Обработка и анализ материалов для написания ВКР | 20.05.2024 |  |
| 3.Написание практического раздела | 25.05.2024 |  |
| 4.Написание аналитического раздела | 31.05.2024 |  |
| 5.Прохождение нормоконтроля | 06.06.2024 |  |
| 6.Предоставление ВКР на рецензию | 08.06.2024 |  |
| 7.Подготовка презентационного материала | 12.06.2024 |  |
| 8.предоставление ВКР на кафедру для допуска к защите | 16.06.2024 |  |

**Содержание**

[Введение 1](#_Toc6294)

[Основные определения связанные с нейронными сетями для решения поставленной задачи 3](#_Toc27743)

[Решение задачи регрессии с помощью алгоритма прогнозировать яркость точки через яркости предыдущих](#_Toc8779)

[16](#_Toc8779)

[Реализация алгоритма прогнозирования яркости точки на основе предыдущих 22](#_Toc11236)

[Сжатие изображения с помощью нейронных сетей методом прогнозирования яркости точки на основе предыдущих 49](#_Toc3950)

[Заключение 67](#_Toc7293)

[Список литературы 69](#_Toc25660)

# 

# Введение

В современном информационном обществе, где огромные объёмы данных требуют высокоточной обработки, нейронные сети становятся неотъемлемой частью технологического прогресса. Нейронные сети представляют собой мощный инструмент в области искусственного интеллекта, эффективно эмулирующий нейронную структуру человеческого мозга. Эта технология нашла широкое применение в различных сферах, включая компьютерное зрение, обработку естественного языка, медицину, финансы и многие другие.

В свою очередь сжатие изображений играет ключевую роль в современном цифровом мире, обеспечивая эффективное хранение и передачу визуальных данных. Традиционные методы, такие как JPEG и PNG, используют различные техники уменьшения избыточности данных для уменьшения размера файлов. Однако с развитием искусственного интеллекта и нейронных сетей появляются новые подходы, которые обещают значительно улучшить качество и эффективность сжатия изображений.

Одним из таких инновационных подходов является метод прогнозирования яркости точки на основе значений предыдущих точек изображения. Эта техника основывается на способности нейронных сетей выявлять и моделировать сложные зависимостей в данных. Идея заключается в том, чтобы предсказать яркость текущей точки, используя информацию о яркости ранее обработанных точек, что позволяет существенно уменьшить количество необходимой для хранения информации без значительных потерь качества.

В данной работе будет рассмотрен метод сжатия изображений с помощью нейронных сетей, основанный на прогнозировании яркости точек через предыдущие. Мы исследуем архитектуру нейронных сетей, подходы к обучению и оптимизации, а также проведём анализ эффективности и качества сжатия в сравнении с традиционными методами(png).

# Основные определения связанные с нейронными сетями для решения поставленной задачи

Машинное обучение опирается на методы из широкого спектра областей, включая информатику, статистику и математику.[17, с 4].

Нейронные сети— математические модели, а также их программные или аппаратные реализации, построенные по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма.[9, с 10]

Другими словами это класс алгоритмов машинного обучения, инспирированный структурой и функционированием нейронов в человеческом мозге. Они состоят из соединённых между собой и взаимодействующих искусственных нейронов, которые образуют слои.

Основные компоненты нейронной сети:

Фундаментальной единицей нейронных сетей является небольшой вычислительный узел, называемый искусственный нейрон. Часто его называют просто нейрон. Искусственные нейроны инспирированы человеческими нейронами, которые являются нервными клетками, составляющими мозг и в значительной степени ответственными за умственные способности человека[7, c 263].

Получается, что нейроны они же узлы это основные единицы обработки в нейронной сети. Они получают входные данные, выполняют определённые вычисления и передают результат на выход.

Математическую формулу нейрона можно выразить как нелинейный сумматор S с помощью уравнения:

, где

n - количество входов нейрона, xi - значения входов нейрона, wi - весовые коэффициенты, f()– нелинейная функция активации, y -выходное значение нейрона.

2.Слои нейронов:

Ключевой концепцией глубокого обучения является многослойный перцептронс. Это разновидность искусственной нейронной сети. Она состоит из нескольких уровней взаимосвязанных узлов, каждый из которых выполняет простые вычисления:[1,с. 55].  
 Входной слой:  
 Входной слой в нейронной сети — это первый слой, который получает и обрабатывает исходные данные. Входной слой просто передаёт входы скрытому слою сети, либо без преобразования, либо выполнив сначала стандартизация входов.[2,с. 124]

Основные характеристики входного слоя:

Размерность данных: Количество нейронов во входном слое соответствует размерности входных данных. Например, если входные данные представляют собой изображение размером 28x28 пикселей, то во входном слое будет 784 нейрона (28 \* 28).

Тип данных: Входной слой принимает данные различных типов, таких как изображения, текст, аудио и др. Тип входных данных определяет структуру и архитектуру последующих слоев нейронной сети.

Предобработка данных: Передача данных в нейронную сеть часто сопровождается предобработкой, которая может включать нормализацию, масштабирование или другие преобразования для улучшения качества обучения сети.

Таким образом, входной слой играет критически важную роль в процессе передачи исходной информации в нейронную сеть для дальнейшей обработки и анализа.

Скрытые слои:

Скрытые слои в нейронной сети — это промежуточные слои, расположенные между входным и выходным слоями. Их основная функция заключается в обработке и преобразовании данных, которые поступают из входного слоя, с целью выявления и изучения сложных и нелинейных зависимостей в данных.

Основные характеристики скрытых слоев:

Количество нейронов: Количество нейронов в каждом скрытом слое может варьироваться в зависимости от задачи и архитектуры сети. Большее количество нейронов позволяет модели улавливать более сложные зависимости, но может привести к переобучению.

Активационные функции: В каждом скрытом нейроне применяется активационная функция, которая помогает моделировать сложные нелинейные зависимости. Примеры активационных функций включают ReLU (Rectified Linear Unit), сигмоидную функцию, гиперболический тангенс (tanh) и др.

Количество слоев: Глубина нейронной сети определяется количеством скрытых слоев. Глубокие нейронные сети (deep neural networks, DNN) имеют много скрытых слоев, что позволяет им извлекать более высокоуровневые признаки из данных.

Обучение и параметры: Во время обучения сети скрытые слои корректируют свои веса и смещения на основе алгоритма оптимизации, такого как градиентный спуск. Это позволяет сети минимизировать ошибку предсказаний и лучше адаптироваться к данным.

Архитектурные особенности: Скрытые слои могут иметь различную структуру в зависимости от типа нейронной сети например: сверточных нейронных сетях (CNN), рекуррентных нейронных сетях (RNN), однослойные сети прямого распространения и многослойные сети прямого распространения.

В многослойной нейронной сети нейроны располагаются по слоям. В простейшем случае в такой сети существует входной слой (input layer) узлов источника, информация от которого передаётся на выходной слой (output layer) нейронов (вычислительные узлы), но не наоборот. Такая сеть называется сетью прямого распространения (feedforward) или ацикличной сетью (acyclic). Такая нейронная сеть называется однослойной (single-layer network), при этом под единственным слоем подразумевается слой вычислительных элементов (нейронов), При подсчёте числа слоёв мы не принимаем во внимание узлы источника, так как они не выполняют никаких вычислений.[4, с. 55]

Многослойные сети прямого распространения (Multilayer Perceptron, MLP) представляют собой разновидность искусственных нейронных сетей, состоящих из множества слоев нейронов, выстроенных в последовательном порядке. В таких сетях каждый нейрон одного слоя соединен с каждым нейроном следующего слоя. Эти сети находят широкое применение в разнообразных задачах, таких как классификация, регрессия и распознавание образов.

Сверточные нейронные сети (CNN) – это специализированные архитектуры, отличающиеся способностью выделять локальные паттерны в данных: на вход им подаются данные произвольного размера, а они выделяют осмысленные локальные паттерны, чувствительные к порядку слов, независимо от того, в каком месте входных данных они встречаются. Они очень хорошо справляются с идентификацией фраз в изъявительном наклонении и идиом заранее ограниченной длины в длинных предложениях или документах.[6,c. 22]

Рекуррентная нейронная сеть (RNN) отличается от сети прямого распространения наличием по крайней мере одной обратной связи. Например, рекуррентная сеть может состоять из единственного слоя нейронов, каждый из которых направляет свой выходной сигнал на входы всех остальных нейронов слоя[4,с. 57].  
 Таким образом, скрытые слои играют ключевую роль в обучении и работе нейронной сети, обеспечивая возможность моделирования сложных зависимостей и паттернов в данных.  
 Выходной слой:

Выходной слой – это последний слой нашей нейронной сети. Если простыми словами это то, где результаты сети взаимодействуют с внешним миром. Когда мы строим нашу архитектуру, то выбираем число нейронов в выходном слое, соответствующее решаемой задаче. Если это задача регрессии с одним единственным численным выходом, то в этом слое будет один нейрон, и величина этого нейрона будет нашим предсказанием.[8,с. 23]

Основные характеристики выходного слоя:

Количество нейронов: Количество нейронов в выходном слое зависит от типа задачи. Например, для задачи классификации на n классов в выходном слое будет n нейронов, каждый из которых соответствует одному классу. Для задачи регрессии выходной слой может содержать один нейрон.

Активационные функции: В выходном слое используются специфические активационные функции в зависимости от задачи:

Для задач классификации часто используется сигмоидная функция, которая преобразует выходные значения в диапазон от 0 до 1, что удобно для интерпретации как вероятность принадлежности к одному из двух классов.

Для задач многоклассовой классификации применяется функция softmax, которая преобразует выходные значения в вероятности, сумма которых равна 1.

В задачах регрессии может использоваться линейная активационная функция, которая позволяет выходным значениям быть произвольными числами.

Формат вывода: Выходной слой формирует результат в виде, подходящем для конкретной задачи. Это могут быть классы, вероятности классов, численные значения и т.д.

Таким образом, выходной слой является ключевым элементом нейронной сети, который преобразует внутренние представления данных в конечный результат, адаптированный под конкретную задачу.

Веса и смещения:

Веса и смещения — это основные параметры нейронной сети, которые регулируются в процессе обучения для минимизации ошибки предсказаний и улучшения точности модели.

Веса (Weights)

Веса — это коэффициенты, которые определяют силу связи между нейронами в различных слоях нейронной сети. Они играют ключевую роль в процессе обучения и функционирования нейронной сети. Веса изменяют значения сигналов, проходящих через нейроны, что позволяет модели адаптироваться к данным и решать поставленную задачу.

Инициализация: Веса обычно инициализируются случайными значениями перед началом обучения.

Обновление: В процессе обучения веса корректируются на основе алгоритма оптимизации (например, градиентного спуска) с использованием градиентов функции потерь.

Роль: Веса определяют, насколько сильно один нейрон влияет на другой. Большие значения весов могут усиливать сигналы, в то время как малые значения могут их подавлять.

Смещения — это дополнительные параметры, которые добавляются к значениям, вычисленным на выходе нейрона, перед применением активационной функции. Они позволяют моделировать более сложные зависимости и улучшать производительность нейронной сети.

Инициализация: Смещения также инициализируются случайными значениями и корректируются в процессе обучения.

Обновление: Смещения обновляются аналогично весам, с использованием алгоритма оптимизации.

Роль: Смещения позволяют нейронам быть активными даже при нулевых входных значениях. Они обеспечивают дополнительную гибкость модели, позволяя ей лучше подстраиваться под данные.

Важность весов и смещений

Обучаемость: Веса и смещения являются основными обучаемыми параметрами нейронной сети. Они настраиваются в процессе обучения для минимизации ошибки предсказаний.

Гибкость: Наличие весов и смещений позволяет моделировать сложные и нелинейные зависимости в данных.

Адаптивность: Корректируя веса и смещения, нейронная сеть адаптируется к различным типам задач и данных, улучшая свою точность и производительность.

Таким образом, веса и смещения являются фундаментальными компонентами нейронных сетей, обеспечивающими их способность обучаться и решать разнообразные задачи.

Функции активации:   
 Функция активации в нейронной сети — это математическая функция, которая применяется к выходным данным нейрона с целью введения нелинейности в модель. Она определяет активацию нейрона, то есть выходное значение, которое передаются следующему слою или используется для принятия решения в задаче.

В качестве активации, в принципе, можно использовать любую (нелинейную) функцию для полносвязной нейронной сети. В течение некоторого времени после их изобретения активации выбирались главным образом на основе их биологической правдоподобности, поскольку именно с этой точки зрения первоначально рассматривались сами нейронные сети. Сегодня функции активации выбираются исходя из практических соображений, включая нашу правильно способность правильно оптимизировать модели, которые их задействуют, а также (разумеется) общую результативность, которую они обеспечивают.[10,с 455].

Мы кратко рассмотрим популярные классические и современные функции активации на нескольких примерах и основные характеристики функций активации.

Основные характеристики функций активации:

Нелинейность: Одна из важнейших характеристик функций активации — их способность внедрять нелинейности в нейронные сети. Это позволяет моделировать сложные и нелинейные зависимости в данных, что часто является ключевым для успешного решения многих задач машинного обучения.

Производная: В процессе обратного распространения ошибки функции активации также играют важную роль, так как необходима их производная для вычисления градиента функции потерь по параметрам сети. Это позволяет корректировать веса и смещения нейронов во время обучения.

Виды функций: Существует множество различных функций активации, каждая из которых обладает своими особенностями и подходит для разных типов задач и архитектур нейронных сетей. Некоторые из наиболее распространенных функций активации включают в себя:

ReLU. Функция активации у них кусочно-линейная:

 [18,с 107].

. Область значений от 0 до бесконечности. Одна из наиболее популярных функций активации, используемая во многих глубоких нейронных сетях.

Leaky ReLU: Вариант ReLU, который позволяет проходить небольшие отрицательные значения. Это помогает избежать проблемы «мертвых» нейронов.  
 ELU  
 Основные характеристики

1.Нелинейность: ELU, как и другие функции активации, вводит нелинейность в нейронную сеть, что позволяет моделировать сложные зависимости в данных.

2. Сглаживание: В отличие от ReLU, которая имеет резкий переход в точке ноль, ELU обеспечивает более плавный переход для отрицательных значений. Это помогает уменьшить проблемы, связанные с градиентами, и улучшить обучение.

3.Отрицательные значения: В отличие от ReLU, ELU может принимать отрицательные значения, что позволяет избежать проблемы «мёртвых» нейронов, когда нейрон перестаёт обновляться из-за того, что его выход постоянно равен нулю.

4.Дифференцируемость: ELU является непрерывной и дифференцируемой функцией, что полезно для методов оптимизации, использующих градиентный спуск.

Преимущества ELU

Быстрая сходимость: ELU может способствовать более быстрой сходимости в процессе обучения по сравнению с ReLU. Улучшенная производительность: За счёт возможности принимать отрицательные значения и сглаженного перехода ELU может улучшать производительность нейронной сети.

Стабильность обучения: ELU помогает стабилизировать процесс обучения, уменьшая риск взрывных градиентов и улучшая общую устойчивость сети.

Функция активации linear (линейная функция активации) — это простейшая форма функции активации, которая просто возвращает входное значение без изменений. Она часто используется в задачах регрессии, где требуется предсказать непрерывное значение.

, где x - входное значение

Основные характеристики линейной функции активации

Линейность: Линейная функция активации сохраняет линейные свойства входных данных. Это означает, что все комбинации линейно зависимых входных данных останутся линейно зависимыми на выходе.

Отсутствие нелинейности: Поскольку линейная функция активации не вводит нелинейности, нейронные сети с линейной активацией на всех уровнях эквивалентны одной линейной модели, независимо от количества слоёв. Это ограничивает способность сети моделировать сложные, нелинейные зависимости.

Дифференцируемость: Линейная функция активации является дифференцируемой, что важно для алгоритмов оптимизации, таких как градиентный спуск.

Преимущества и недостатки линейной функции активации

Преимущества:

Простота: Линейная функция активации проста и легко интерпретируема.

Подходит для регрессии: Она полезна в задачах регрессии, где модель должна предсказывать непрерывные значения.

Недостатки:

Ограниченные возможности моделирования: Поскольку линейная активация не может моделировать нелинейные зависимости, она не подходит для задач, требующих сложного анализа данных.

Отсутствие способности к обучению сложных функций: Нейронные сети с линейной активацией не могут эффективно обучаться сложным функциям, что ограничивает их применимость в более сложных задачах.

Таким образом, линейная функция активации играет важную роль в задачах регрессии, обеспечивая возможность предсказания непрерывных значений, но она не подходит для задач, требующих моделирования сложных, нелинейных зависимостей.

Оптимизатор в нейронных сетях

В контексте нейронных сетей, оптимизатор – это алгоритм, который определяет способ обновления параметров модели в процессе обучения с целью минимизации функции потерь. Функция потерь (или целевая функция) измеряет разницу между предсказанными значениями модели и их истинными значениями. Оптимизаторы играют ключевую роль в оптимизации модели и настройке её параметров для достижения лучшей производительности.

Основные определения, связанные с оптимизаторами:

1.Функция потерь (Loss Function): Функция потерь определяет величину ошибки или расхождения между предсказанными значениями модели и их истинными значениями. Она является мерой того, насколько хорошо модель справляется с поставленной задачей. Цель обучения нейронной сети заключается в минимизации значения функции потерь.

2.Градиент (Gradient): Градиент функции потерь по параметрам модели указывает направление наибольшего убывания функции. Оптимизаторы используют градиент для определения, какие параметры модели нужно изменить и в каком направлении, чтобы уменьшить функцию потерь.

3.Скорость обучения (Learning Rate): Скорость обучения определяет размер шага, который оптимизатор делает при обновлении параметров модели. Это важный гиперпараметр, который влияет на скорость сходимости обучения и качество модели. Слишком большая скорость обучения может привести к расхождению, а слишком маленькая – к медленной сходимости.

4.Адаптивная скорость обучения (Adaptive Learning Rate): Некоторые оптимизаторы, такие как Adam, RMSprop и Adagrad, автоматически адаптируют скорость обучения для каждого параметра на основе истории градиентов. Это позволяет более эффективно настраивать параметры модели и ускоряет процесс обучения.

5.Батч (Batch): Группа обучающих примеров, которые используются для вычисления градиента функции потерь. Обычно обучение проводится на мини-батчах, чтобы ускорить процесс обучения и сгладить градиенты.

Применение оптимизатора в решении задачи:

В контексте решения поставленной задачи – регрессии для предсказания яркости точек на основе предыдущих значений – выбор оптимизатора имеет решающее значение для эффективного обучения модели. Некоторые оптимизаторы, такие как Adam, SGD (стохастический градиентный спуск) и RMSprop, часто применяются в задачах регрессии изображений.

Последний алгоритм, который мы рассмотрим, называется Adam (adaptive moment estimation, адаптивное оценивание импульса), Он объединяет идеи алгоритмов RMSProp и Momentum в одном оптимизаторе. Как и Momentum, он использует экспоненциально взвешенные средние прошлых производных, и как RMSProp, он использует экспоненциально взвешенные средние прошлых квадратичных производных.[11,с 170] Простыми словами Adam этот оптимизатор адаптирует скорость обучения для каждого параметра на основе первого и второго моментов градиента. Adam обычно демонстрирует хорошую производительность и сходимость во многих задачах обучения нейронных сетей.

SGD (Stochastic Gradient Descent): Это классический оптимизатор, который обновляет параметры модели в направлении, противоположном градиенту функции потерь. Он является одним из самых простых и понятных оптимизаторов, но его использование может потребовать тщательной настройки скорости обучения.

RMSprop (Root Mean Square Propagation) - это адаптивный оптимизатор, который используется для обучения нейронных сетей. Он представляет собой модификацию метода градиентного спуска, который адаптирует скорость обучения для каждого параметра модели.

Задачи машинного обучения

Поговорим о типовых задачах машинного обучения. Стандартно

задачи машинного обучения разделяют на следующие:

1. обучение с учителем (Supervised Learning);

2. обучение без учителя (Unsupervised Learning);

3. обучение с подкреплением (Reinforcement Learning).

Для обучения с учителем характерно использование обучающего набора данных (тренировочной выборки). Для каждого экземпляра из набора данных есть пары «входные данные/признаки — ожидаемый ответ». В этом случае задачей является поиск модели или алгоритма, который предсказывает ожидаемые целевые ответы.[12,с 19].

Обучение без учителя предполагает использование данных для моделирования, поступающих без соответствующих категорий и обработки.При достаточном объёме данных можно находить паттерны и структур­ы.[13,с 43].

Обучение с подкреплением позволяет обучать машину, не подсказывая ей ожидаемого ответа, а только сообщая, является ли конкретный ответ правильным или нет. Оно используется, когда мы можем оценить качество ответа системы без возможности предоставить ей этот правильный ответ[15,с 300]

Еще одним основным понятие является функция потерь.  
Рассмотрим метку yn для конкретного примера и соответствующий ей прогноз, который мы делаем на основе xn. Чтобы определить, что такое «хорошо подстраиваться к данным», необходимо определить функцию потерь, которая принимает в качестве ввода метку базовой истины и прогноз, а в ответ выдает неотрицательное число (именуемое «потерей»), отражающее, какова величина ошибки, допущенной нами при этом конкретном прогнозе. Наша целью при нахождении хорошего вектора параметров — минимизировать среднюю потерю 1 на множестве из N учебных примеров[16, с 329].

# Решение задачи регрессии с помощью алгоритма прогнозировать яркость точки через яркости предыдущих

Регрессия является одной из ключевых задач в области машинного обучения, где модель стремится предсказать непрерывные числовые значения.[5,с. 6].Традиционно для решения задач регрессии использовались такие методы, как линейная регрессия, полиномиальная регрессия и другие. Однако с развитием технологий искусственного интеллекта и нейронных сетей стало возможным решать задачи регрессии с использованием более сложных моделей, способных обрабатывать большие объёмы данных и извлекать из них скрытые зависимости.  
 Давайте сформулируем задачу.Мы хотим предсказывать яркость определённой точки изображения на основе яркости соседних точек. Пусть у нас имеется матрица яркости, которая может быть либо одной из RGB - компонент цветного изображения, либо яркостью серого для черно-белого изображения. Используя эту матрицу, будем строить обучающую матрицу для задачи регрессии.

Формирование обучающей матрицы

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | -3 | -2 | -1 | 0 | 1 | 2 | 3 |
| -3: |  |  | 15 | 13 | 16 |  |  |
| -2: |  | 10 | 7 | 5 | 8 | 11 |  |
| -1: | 14 | 6 | 2 | 1 | 3 | 9 | 17 |
| 0: | 12 | 4 | 0 | X | - | - | - |

Мы хотим выразить яркость точки X через яркости предыдущих точек. Таким образом, целевая функция примет следующий вид:



Где точки определяются как:









…





Если изображение имеет размеры , то будем брать индексы  в пределах:



Преобразование данных для нейронной сети таким образом, из каждой цветовой компоненты изображения размера мы получаем  строк обучающей матрицы, в каждой строке в начале записывается яркость точки, а затем яркости предыдущих 18 точке в указанном выше порядке.

Для применения в нейронных сетях сделаем ещё одно преобразование. В каждой строе матрицы вычтем из всех её элементов второй, то есть . На результатах работы нейронной сети это никак не скажется, но используемые числа будут меньше. В результате второй столбец обучающей матрицы будет состоять из одних нулей и его можно исключить,

Построенную таким образом обучающую матрицу будем подавать на вход нейронной сети. Эта матрица в виде текстового файла будет выглядеть так

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| y | X0 | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 | X9 | X10 | X11 | X12 | X13 | X14 | X15 | X16 |
| 0, 0, -1, 1, -1,  … | 0, 0,  -1, 1,  -1,  … | 0, 0,  -1, 1,  -1,  … | 0, 0,  -1, 1,  -1,  … | 0, 0,  -1, 1,  -1,  … | 0, 0,  -1, 1,  -1,  … | 0, 0,  -1, 1,  -1,  … | 0, 0,  -1, 1,  -1,  … | 0, 0,  -1, 1,  -1,  … | 0, 0,  -1, 1,  -1,  … | 0, 0,  -1, 1,  -1,  … | 0, 0,  -1, 1,  -1, -  … | 0, 0,  -1, 1,  -1,  … | 0, 0,  -1, 1,  -1,  … | 0, 0,  -1, 1,  -1,  … | 0, 0,  -1, 1,  -1,  … | 0, 0,  -1, 1,  -1,  … | 0, 0,  -1, 1,  -1,  … |

Обучающие и тестовые данные

У нас есть два файла с обучающими и тестовыми данными: Каждая строка в этих матрицах представляет собой набор значений, где первый столбец - это прогнозируемая переменная (y), а последующие 17 столбцов содержат значения (x0, x1, ..., x16).

Цель задачи

Построить модель, которая сможет предсказать значение y на основе значений x0, x1, ..., x16. Математически это выражается как:



Прогноз должен быть как можно более точным, и точность измеряется с использованием среднеквадратичного отклонения (Mean Squared Error, MSE). MSE считается для каждой строки данных, и затем общее MSE дает представление о том, насколько хорошо модель обобщается на обучающих данных. 

Используемые данные представляют собой наборы признаков (x0, x1, ..., x16) и соответствующих им прогнозируемых значений (y). В процессе построения модели машинного обучения, такой как линейная регрессия или нейронная сеть, мы подстраиваем параметры модели таким образом, чтобы минимизировать RMSE на обучающих данных.

Почему лучше брать RMSE чем MSE

RMSE имеет несколько преимуществ перед MSE, которые делают его предпочтительным в некоторых ситуациях:

Сравнение с исходной шкалой:

RMSE выражается в тех же единицах, что и прогнозируемая переменная (y). Это делает интерпретацию ошибки более интуитивной и понятной, поскольку она сопоставима с величинами входных данных.

Например, если предсказываемая переменная — это яркость точки изображения, измеряемая в диапазоне от 0 до 255, RMSE также будет выражаться в этом диапазоне. MSE, напротив, выражается в квадратных единицах, что может быть менее наглядным и сложным для интерпретации.

Влияние больших ошибок:

RMSE, как и MSE, чувствителен к большим ошибкам. Однако, поскольку RMSE берёт квадратный корень из MSE, он уменьшает влияние крупных ошибок на итоговую оценку по сравнению с MSE, где ошибки возводятся в квадрат и суммируются. Это делает RMSE более устойчивым к выбросам по сравнению с MSE.

Понятность и интерпретация:

RMSE легче интерпретировать в контексте проблемы. Например, если RMSE составляет 10 для предсказания яркости точки изображения, это означает, что средняя ошибка предсказания модели составляет 10 единиц яркости. В случае MSE ошибка будет выражаться в квадратных единицах, что может быть менее интуитивно.

Сопоставимость между различными моделями и задачами:

Поскольку RMSE выражается в тех же единицах, что и прогнозируемая переменная, его легче использовать для сравнения между различными моделями и задачами. Например, если у вас есть две модели для предсказания яркости точки изображения, RMSE даст вам более наглядное сравнение точности моделей, чем MSE.

Пример использования RMSE в задаче

Рассмотрим пример, где мы хотим предсказывать яркость точки изображения. Пусть у нас есть обучающая выборка с реальными значениями яркости и предсказанными значениями модели:

|  |  |
| --- | --- |
| Реальные значения y | Предсказынне значения y |
| 50 | 48 |
| 150 | 155 |
| 200 | 190 |
| 250 | 240 |

Для вычисления RMSE сначала вычислим ошибки предсказания, затем возведем их в квадрат, найдем среднее значение и извлечем квадратный корень:

Вычисляем ошибки:

e1 = 50 − 48 = 2

e2= 150 − 155 = −5

e3= 200 − 190 = 10

e4= 250 − 240 =10

Возводим ошибки в квадрат и находим среднее:



Среднее значение: MSE = 

Извлекаем квадратный корень:



Таким образом, RMSE составляет 7.57, что означает, что средняя ошибка предсказания модели составляет примерно 7.57 единиц яркости, что легко интерпретировать и сравнить с диапазоном значений яркости в изображении (от 0 до 255).

Использование RMSE вместо MSE в задачах регрессии часто оказывается более полезным и интуитивно понятным, поскольку RMSE выражается в тех же единицах, что и прогнозируемая переменная, и позволяет легко интерпретировать и сравнивать результаты между различными моделями и задачами.

Итог

Наша цель - создать модель, которая наилучшим образом учитывает взаимосвязь между входными признаками (x0, x1, ..., x16) и прогнозируемой переменной (y), чтобы предсказывать y с минимальной ошибкой.

# Реализация алгоритма прогнозирования яркости точки на основе предыдущих

Мы будем использовать язык программирования Python. С него удобно начинать, поскольку он прост в изучении. Инструкции, на­ писанные на Python одними людьми, легко читают и понимают другими людьми[3,с. 129].

Кроме этого python имеет большое количество преимуществ таких как:

Python предоставляет богатый набор библиотек для машинного обучения:

Scikit-learn: Прекрасная библиотека для выполнения задач машинного обучения, включая регрессию, классификацию, кластеризацию и оценку моделей.

TensorFlow и Keras: Мощные библиотеки для создания и обучения нейронных сетей.

Кроме того, для обработки данных Python предлагает:

Pandas: библиотека Python для обработки и анализа данных. Она построена на основе структуры данных, называемой DataFrame и смоделированной по принципу датафреймов среды статистического программирования. Проще говоря, DataFrame библиотеки pandas представляет собой таблицу, похожую на электронную таблицу Excel[13,с 23].

NumPy – это один из основных пакетов для научных вычислений в Python. Он содержит функциональные возможности для работы с многомерными массивами, высокоуровневыми математическими функциями[13, с 20].

2. Простота и читаемость кода

Python известен своей простотой и читаемостью. Его синтаксис позволяет писать четкий и понятный код, что особенно важно при работе с алгоритмами машинного обучения, где сложность моделей и данных может быть высокой.

3. Активное сообщество и поддержка

Python имеет одно из крупнейших и самых активных сообществ разработчиков. Это означает, что для большинства задач можно найти уже готовые решения, а также обширную документацию и примеры использования. Форумы и сайты, такие как Stack Overflow, предлагают множество ресурсов для помощи в решении возникающих проблем.

4. Легкость интеграции

Python легко интегрируется с другими языками и технологиями. Например, он поддерживает интеграцию с C/C++, что позволяет использовать высокопроизводительные вычислительные библиотеки. Python также хорошо работает с различными базами данных и веб-сервисами, что делает его универсальным инструментом для разработки и деплоя моделей машинного обучения.

5. Мощные инструменты для визуализации данных

Визуализация данных играет ключевую роль в понимании и анализе данных. Python предлагает множество инструментов для визуализации, таких как:

Matplotlib: Для создания статичных, анимационных и интерактивных графиков.

Python является идеальным выбором для реализации алгоритма прогнозирования яркости точки на основе предыдущих значений благодаря своему богатому набору библиотек, простоте и читаемости кода, активному сообществу, лёгкости интеграции, мощным инструментам для визуализации данных и поддержке работы с большими данными. Эти преимущества делают его одним из наиболее популярных языков программирования в области машинного обучения и обработки данных.

Для начала получим из цветного изображение серое это делается следующим образом:

import numpy as np  
from PIL import Image  
def a\_gray(a):  
 y = np.round(0.299 \* a[:, :, 0] + 0.587 \* a[:, :, 1] + 0.114 \* a[:, :, 2])  
 return np.array(y, dtype=int)  
def save\_gray\_image(image\_array, output\_path):  
 img = Image.fromarray(np.uint8(image\_array), 'L')  
 img.save(output\_path)  
input\_image = 'pic.png'  
output\_gray\_image = 'picgray.png'  
img\_data = Image.open(input\_image)  
img\_arr = np.array(img\_data)  
gray\_img\_arr = a\_gray(img\_arr)  
save\_gray\_image(gray\_img\_arr, output\_gray\_image)

Это программа делает следующее:

Импортируемые библиотеки

import numpy as np

from PIL import Image

1.numpy — это библиотека для работы с массивами и матрицами, а также для выполнения математических операций над ними.

2.PIL (Python Imaging Library), а точнее её модуль Image из Pillow (современная версия PIL), используется для работы с изображениями.

Функция для преобразования цветного изображения в серое

def a\_gray(a): y = np.round(0.299 \* a[:, :, 0] + 0.587 \* a[:, :, 1] + 0.114 \* a[:, :, 2]) return np.array(y, dtype=int)

Функция a\_gray выполняет следующие действия:

1.Принимает трехмерный массив a, который представляет собой цветное изображение в формате RGB.

a[:, :, 0] — это красный (Red) канал изображения.

a[:, :, 1] — это зеленый (Green) канал изображения.

a[:, :, 2] — это синий (Blue) канал изображения.

2.Преобразует цветное изображение в серое, используя формулу для яркости:

0.299 \* R + 0.587 \* G + 0.114 \* B

Эти коэффициенты соответствуют восприятию различных цветов человеческим глазом.

3.Округляет значения до ближайшего целого с помощью np.round.

Преобразует результат в массив целых чисел типа int и возвращает его.

Функция для сохранения серого изображения

def save\_gray\_image(image\_array, output\_path):

img=Image.fromarray(np.uint8(image\_array), 'L')

img.save(output\_path)

Функция save\_gray\_image выполняет следующие действия:

1.Принимает массив изображения image\_array и путь для сохранения output\_path.

2.Преобразует массив в изображение формата PIL с помощью Image.fromarray.

np.uint8 преобразует массив к типу uint8 (8-битные беззнаковые целые числа).

'L' указывает на то, что изображение будет в оттенках серого.

3.Сохраняет изображение по указанному пути с помощью метода save.

Основной блок кода

input\_image = 'pic.png'

output\_gray\_image = 'picgray.png'

img\_data = Image.open(input\_image)

img\_arr = np.array(img\_data)

gray\_img\_arr = a\_gray(img\_arr)

save\_gray\_image(gray\_img\_arr, output\_gray\_image)

Этот блок кода выполняет следующие действия:

1.Задает путь к входному цветному изображению input\_image и выходному серому изображению output\_gray\_image.

2.Загружает цветное изображение с помощью Image.open:

img\_data = Image.open(input\_image)

3.Преобразует изображение в массив numpy:

img\_arr = np.array(img\_data)

4.Преобразует цветное изображение в серое, используя функцию a\_gray:

gray\_img\_arr = a\_gray(img\_arr)

5.Сохраняет полученное серое изображение с помощью функции save\_gray\_image:

save\_gray\_image(gray\_img\_arr, output\_gray\_image)

Таким образом, этот код загружает цветное изображение, преобразует его в серое и сохраняет полученное серое изображение.

Затем получим обучающую матрицу для этого будем использовать модифицированную программу:

import numpy as np  
from PIL import Image  
np.set\_printoptions(precision=6, linewidth=120, suppress=True)  
def a\_gray(a):  
 y = np.round(0.299\*a[:, :, 0] + 0.587\*a[:, :, 1] + 0.114\*a[:, :, 2])  
 return np.array(y, dtype=int)  
def save\_gray\_image(image\_array, output\_path):  
 img = Image.fromarray(np.uint8(image\_array), 'L')  
 img.save(output\_path)  
def tteach18(a):  
 step\_y = 2  
 step\_x = 2  
 my, mx = a.shape  
 b = np.zeros(((my - 3) \* (mx - 6), 19), dtype=np.uint8)  
 i = 0  
 for y in range(3, my, step\_y):  
 for x in range(3, mx - 3, step\_x):  
 b[i, 0] = a[y, x]  
 b[i, 1] = a[y, x - 1]  
 b[i, 2] = a[y - 1, x]  
 b[i, 3] = a[y - 1, x - 1]  
 b[i, 4] = a[y - 1, x + 1]  
 b[i, 5] = a[y, x - 2]  
 b[i, 6] = a[y - 2, x]  
 b[i, 7] = a[y - 1, x - 2]  
 b[i, 8] = a[y - 2, x - 1]  
 b[i, 9] = a[y - 2, x + 1]  
 b[i, 10] = a[y - 1, x + 2]  
 b[i, 11] = a[y - 2, x - 2]  
 b[i, 12] = a[y - 2, x + 2]  
 b[i, 13] = a[y, x - 3]  
 b[i, 14] = a[y - 3, x]  
 b[i, 15] = a[y - 1, x - 3]  
 b[i, 16] = a[y - 3, x - 1]  
 b[i, 17] = a[y - 3, x + 1]  
 b[i, 18] = a[y - 1, x + 3]  
 i += 1  
 return b[:i]  
def teach17(in\_name, rgb='rgb'):  
 img\_data = Image.open(in\_name)  
 if rgb == 'y':  
 img\_data = img\_data.convert('L')  
 img\_arr = np.array(img\_data)  
 res = tteach18(img\_arr)  
 else:  
 img\_arr = np.array(img\_data)  
 if rgb == 'rgb':  
 ar = tteach18(img\_arr[:, :, 0]); print('r')  
 ag = tteach18(img\_arr[:, :, 1]); print('g')  
 ab = tteach18(img\_arr[:, :, 2]); print('b')  
 res = np.vstack((ar, ag, ab))  
 else:  
 if rgb == 'r':  
 a = img\_arr[:, :, 0]  
 elif rgb == 'g':  
 a = img\_arr[:, :, 1]  
 elif rgb == 'b':  
 a = img\_arr[:, :, 2]  
 else:  
 raise Exception("teach17: Illegal type <%s>" % rgb)  
 res = tteach18(a)  
 return res  
input\_image = 'isaak.png'  
output\_gray\_image = 'isaakgray.png'  
img\_data = Image.open(input\_image)  
img\_arr = np.array(img\_data)  
gray\_img\_arr = a\_gray(img\_arr)  
save\_gray\_image(gray\_img\_arr, output\_gray\_image)  
A = teach17(input\_image, rgb='y')  
print('matrix.shape=', A.shape)  
header = ' y, x0, x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12, x13, x14, x15, x16, x17'  
np.savetxt(r'test.csv', A, delimiter=',', fmt='%3d', header=header, comments='')

Из нового здесь:

Настройка вывода numpy: np.set\_printoptions настраивает формат вывода массивов numpy, устанавливая точность до 6 знаков после запятой, ширину строки до 120 символов и подавляя научную нотацию для малых чисел.

Функция создания обучающей матрицы

def tteach18(a):

step\_y = 2

step\_x = 2

my, mx = a.shape

b = np.zeros(((my - 3) \* (mx - 6), 19), dtype=np.uint8)

i = 0

for y in range(3, my, step\_y):

for x in range(3, mx - 3, step\_x):

b[i, 0] = a[y, x]

b[i, 1] = a[y, x - 1]

b[i, 2] = a[y - 1, x]

b[i, 3] = a[y - 1, x - 1]

b[i, 4] = a[y - 1, x + 1]

b[i, 5] = a[y, x - 2]

b[i, 6] = a[y - 2, x]

b[i, 7] = a[y - 1, x - 2]

b[i, 8] = a[y - 2, x - 1]

b[i, 9] = a[y - 2, x + 1]

b[i, 10] = a[y - 1, x + 2]

b[i, 11] = a[y - 2, x - 2]

b[i, 12] = a[y - 2, x + 2]

b[i, 13] = a[y, x - 3]

b[i, 14] = a[y - 3, x]

b[i, 15] = a[y - 1, x - 3]

b[i, 16] = a[y - 3, x - 1]

b[i, 17] = a[y - 3, x + 1]

b[i, 18] = a[y - 1, x + 3]

i += 1

return b[:i]

1.tteach18: Создает обучающую матрицу из серого изображения. Пробегает по изображению с шагом 2 по осям x и y, формируя векторы размером 19.

2.Заполнение матрицы: Заполняет матрицу b, где каждый вектор состоит из центрального пикселя и соседних пикселей.

Функция обработки изображения

def teach17(in\_name, rgb='rgb'):

img\_data = Image.open(in\_name)

if rgb == 'y':

img\_data = img\_data.convert('L') # Конвертируем изображение в серый формат

img\_arr = np.array(img\_data)

res = tteach18(img\_arr)

else:

img\_arr = np.array(img\_data)

if rgb == 'rgb':

ar = tteach18(img\_arr[:, :, 0]); print('r')

ag = tteach18(img\_arr[:, :, 1]); print('g')

ab = tteach18(img\_arr[:, :, 2]); print('b')

res = np.vstack((ar, ag, ab))

else:

if rgb == 'r':

a = img\_arr[:,:,0]

elif rgb == 'g':

a = img\_arr[:,:,1]

elif rgb =='b':

a = img\_arr[:, :, 2]

else:

raise Exception("teach17: Illegal type <%s>" % rgb)

res = tteach18(a)

return res

teach17: Обрабатывает изображение в зависимости от заданного параметра rgb.

Если rgb='y', изображение конвертируется в серый формат и обрабатывается.

Если rgb='rgb', обрабатываются три цветовых канала по отдельности, создавая три матрицы обучающих данных, которые затем объединяются.

Если rgb указывает на конкретный канал (r, g или b), обрабатывается только этот канал.

Основной код

input\_image = 'isaak.png'

output\_gray\_image = 'isaakgray.png'

img\_data = Image.open(input\_image)

img\_arr = np.array(img\_data)

gray\_img\_arr = a\_gray(img\_arr)

save\_gray\_image(gray\_img\_arr, output\_gray\_image)

A = teach17(input\_image, rgb='y')

print('matrix.shape=', A.shape)

header = ' y, x0, x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12, x13, x14, x15, x16, x17'

np.savetxt(r'test.csv', A, delimiter=',', fmt='%3d', header=header, comments='')

Задает путь к входному изображению input\_image и выходному серому изображению output\_gray\_image.

Загружает входное изображение в img\_data.

Преобразует цветное изображение в серое с помощью a\_gray и сохраняет его с помощью save\_gray\_image.

Создает обучающую матрицу A для серого изображения, вызвав teach17 с параметром rgb='y'.

Печатает форму матрицы A.

Сохраняет обучающую матрицу в CSV файл test.csv с заголовком header.

Мы хотим сравнивать полученные значения с линейной регрессией

Для решения линейной регрессии будем использовать следующую программу:

def lin\_regr(A):  
 x = A[:, 1:]  
 y = A[:, 0]  
 lsq = np.linalg.lstsq(x, y, rcond=None)  
 di = x @ lsq[0] - y  
 su2 = np.sum(di\*di)/len(y)  
 print('mse =%12.8f' %su2)  
 print('w=', lsq[0])

Функция lin\_regr делает следующие:

1)Принимает матрицу A в качестве аргумента.

2)Извлекает значения второго и последующих столбцов матрицы A в переменную x.

3)Извлекает значения первого столбца матрицы A в переменную y.

4)Использует функцию np.linalg.lstsq для выполнения метода наименьших квадратов, находя коэффициенты линейного уравнения, выражающего 0-й столбец через остальные.

5)Вычисляет вектор невязок di как разность между предсказанными и фактическими значениями.

6)Вычисляет среднеквадратичную ошибку (MSE) как сумму квадратов невязок, деленную на количество элементов вектора y.

7)Выводит значение MSE на экран.

Для задачи прогнозирования значений с помощью нейронных сетей будем изначально использовать такую программ по ходу работы дополняя и улучшая её(уменьшая RMSE):

import pandas as pd  
from keras.models import Sequential  
from keras.layers import Dense  
import numpy as np  
data = pd.read\_csv("isaak.csv")  
X = data.iloc[:,1:]  
y = data.iloc[:,0]  
def lin\_regr(A): # линейно выражаем 0-й столбец матрицы A через остальные.  
 ''' возвращаем MSE, вектор коэффициентоы w[i] '''  
 x = A[:, 1:]  
 y = A[:, 0]  
 lsq = np.linalg.lstsq(x, y, rcond=None)  
 di = x @ lsq[0] - y  
 su2 = np.sum(di\*di)/len(y)  
 print('mse=%12.8f' %su2)  
 print('w=', lsq[0])  
lin\_regr(data.values)  
model = Sequential()  
model.add(Dense(units=128, activation='relu'))  
model.add(Dense(units=64, activation='relu'))  
model.add(Dense(units=32, activation='relu'))  
model.add(Dense(units=1, activation='linear'))  
model.compile(loss='mean\_squared\_error', optimizer='adam')  
model.fit(X, y, epochs=20, batch\_size=128)  
predictions = model.predict(X)  
mse = ((predictions[:,0] - y)\*\*2).mean()  
print("Mean squared error: {}".format(mse))  
print(model.summary())  
Данная программа выполняет следующие действия:

1)Импорт библиотек:

1.1)import pandas as pd: Импорт библиотеки pandas с псевдонимом pd для работы с данными в формате DataFrame.

1.2)from keras.models import Sequential: Импорт класса Sequential из библиотеки Keras для создания последовательной модели нейронной сети.

1.3)from keras.layers import Dense: Импорт класса Dense из библиотеки Keras для добавления полносвязных слоев к модели.

1.4)import numpy as np: Импорт библиотеки numpy с псевдонимом np для работы с массивами и матрицами.

2)Чтение данных:

2.1)data = pd.read\_csv("isaak.csv"): Загрузка данных из файла "isaak.csv" в объект DataFrame с использованием pandas.

3)Подготовка данных для линейной регрессии:

3.1)X = data.iloc[:, 1:]: Выделение всех столбцов, кроме первого, в матрицу признаков X.

3.2)y = data.iloc[:, 0]: Выделение первого столбца вектора целевой переменной y.

4)Линейная регрессия:

4.1)Определение функции lin\_regr(A), которая принимает матрицу A и линейно выражает 0-й столбец через остальные, выводя среднеквадратичную ошибку (MSE) и вектор коэффициентов w.

4.2)Вычисление MSE и вывод вектора коэффициентов w.

5)Нейронная сеть с использованием Keras:

5.1)Создание объекта модели Sequential.

5.2)Добавление полносвязных слоев с активационной функцией ReLU.

5.3)Добавление полно связного слоя с активационной функцией linear.

5.4)Компиляция модели с использованием среднеквадратичной ошибки в качестве функции потерь и оптимизатора Adam.

5.5)Обучение модели на данных X и y в течение 20 эпох с размером пакета 128.

batch\_size: это количество примеров данных, которые одновременно подаются в нейронную сеть для обработки на каждом шаге обучения. В данной программе в процессе каждой эпохи модель обучается на данных, разбитых на пакеты по 128 примеров.

5.6)Получение предсказаний модели на данных X.

6)Вычисление MSE для нейронной сети:

6.1)Вычисление MSE между предсказаниями модели и истинными значениями.

7)Вывод результатов:

7.1)Вывод среднеквадратичной ошибки для линейной регрессии и нейронной сети.

7.2)Вывод общей информации о модели нейронной сети.

В результате выполнения программы вы увидите выводы о среднеквадратичной ошибке для линейной регрессии и нейронной сети, а также краткую информацию о параметрах модели нейронной сети.

Таким образом, данная программа реализует сравнение линейной регрессии и нейронной сети в задаче регрессии на входных данных из файла "isaak.csv".

Далее приведён пример работы программы для прогнозирования значения y (Обучение и тестирование производится на серой картинке):  


Рисунок 1 Изначальное изображение

Для начала мы хотим конвертировать цветное изображение в серое и получить обучающую матрицу для серого изображения.



Рисунок 2 Серое изображение

Обучающая матрица получается такая:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| y | X0 | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 | X9 | X10 | X11 | X12 | X13 | X14 | X15 | X16 | X17 |
| 238 | 236 | 236 | 235 | 237 | 235 | 235 | 237 | 237 | 234 | 236 | 234 | 235 | 235 | 234 | 237 | 237 | 235 | 236 |
| 239 | 238 | 237 | 237 | 237 | 236 | 236 | 237 | 236 | 236 | 237 | 237 | 238 | 235 | 236 | 237 | 235 | 239 | 237 |
| 240 | 238 | 238 | 237 | 239 | 238 | 238 | 239 | 237 | 237 | 239 | 240 | 238 | 238 | 240 | 237 | 237 | 239 | 240 |
| 238 | 240 | 240 | 239 | 240 | 239 | 240 | 240 | 241 | 239 | 238 | 241 | 238 | 240 | 237 | 239 | 241 | 234 | 240 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

Размер матрицы (165004, 19)

(разрежения изображения 1000\*666, т.е количество обучающих векторов 1000\*666\*3 = 1998000)

Результаты тестирования следующие:

Для линейной регрессии

mse= 96.86823634

w= [ 7.80955518e-01 5.15633168e-01 -3.70413175e-01 8.14556033e-02

-1.24382900e-01 -9.66463467e-03 3.51003801e-02 1.58474899e-07

-2.31498753e-02 2.63358528e-02 1.40998598e-02 -4.95136323e-03

1.06436872e-01 8.01657248e-02 -7.82971579e-02 -4.46927049e-02

-1.64634417e-02 3.13910671e-02]

С помощью нейронных сетей

...

Epoch 17/20

1290/1290 ━━━━━━━━━━━━━━━━━━━━ 1s 914us/step - loss: 82.4537

Epoch 18/20

1290/1290 ━━━━━━━━━━━━━━━━━━━━ 1s 936us/step - loss: 83.8706

Epoch 19/20

1290/1290 ━━━━━━━━━━━━━━━━━━━━ 1s 775us/step - loss: 81.5175

Epoch 20/20

1290/1290 ━━━━━━━━━━━━━━━━━━━━ 1s 951us/step - loss: 84.0122

5157/5157 ━━━━━━━━━━━━━━━━━━━━ 4s 681us/step

Mean squared error: 78.80063985533026

RMSE для линейной регрессии 9.8 с помощью нейронных сетей 8.83

Вывод что нейронная сеть справляется лучше

Попробуем улучшить значение mse для нейронных сетей путем добавления ещё одного слоя, изменения количества нейронов в слоях, изменения batch\_size = 512 и увелечения количества epoch.

Программа будет выглядит следующим образом:

import pandas as pd  
from keras.models import Sequential [20]  
from keras.layers import Dense [20]  
import numpy as np  
data = pd.read\_csv("isaak.csv")  
X = data.iloc[:,1:]  
y = data.iloc[:,0]  
def lin\_regr(A):  
 x = A[:, 1:]  
 y = A[:, 0]  
 lsq = np.linalg.lstsq(x, y, rcond=None)  
 di = x @ lsq[0] - y  
 su2 = np.sum(di\*di)/len(y)  
 print('mse=%12.8f' %su2)  
 print('w=', lsq[0])  
lin\_regr(data.values)  
model = Sequential()  
model.add(Dense(units=512, activation='relu'))  
model.add(Dense(units=256, activation='relu'))  
model.add(Dense(units=128, activation='relu'))  
model.add(Dense(units=64, activation='relu'))  
model.add(Dense(units=1, activation='linear'))  
model.compile(loss='mean\_squared\_error', optimizer=’adam ’)  
model.fit(X, y, epochs=30, batch\_size=512)  
predictions = model.predict(X)  
mse = ((predictions[: ,0] - y)\*\*2).mean()  
print("Mean squared error: {}".format(mse))  
print(model.summary())

mse= 96.86823634

w= [ 7.80955518e-01 5.15633168e-01 -3.70413175e-01 8.14556033e-02

-1.24382900e-01 -9.66463467e-03 3.51003801e-02 1.58474899e-07

-2.31498753e-02 2.63358528e-02 1.40998598e-02 -4.95136323e-03

1.06436872e-01 8.01657248e-02 -7.82971579e-02 -4.46927049e-02

-1.64634417e-02 3.13910671e-02]

Epoch 26/30

323/323 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 79.8419

Epoch 27/30

323/323 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 75.8325

Epoch 28/30

323/323 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 76.3971

Epoch 29/30

323/323 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 76.8339

Epoch 30/30

323/323 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 75.7147

5157/5157 ━━━━━━━━━━━━━━━━━━━━ 5s 986us/step

Mean squared error: 73.0590166727416

Количество обучаемых параметров:

|  |  |  |
| --- | --- | --- |
| Layer (type) | Output Shape | Param # |
| dense (Dense) | (None, 512) | 9728 |
| dense\_1 (Dense) | (None, 256) | 131328 |
| dense\_2 (Dense) | (None, 128) | 32896 |
| dense\_3 (Dense) | (None, 64) | 8256 |
| dense\_4 (Dense) | (None, 1) | 65 |

Total params: 546,821

Trainable params: 182,273

Non-trainable params: 0

Optimizer params: 364,548

Линейной Rmse = 9.85

С помощью сетей Rmse = 8.5

В предыдущей версии получилось Rmse = 8.83 в новой Rmse = 8.5

Количество параметров изменилось с 12,801 на 182,273

Попробуем изменить количество нейронов в сети и batch\_size:

model.add(Dense(units=1024, activation='relu'))  
model.add(Dense(units=512, activation='relu'))  
model.add(Dense(units=256, activation='relu'))  
model.add(Dense(units=128, activation='relu'))  
model.add(Dense(units=1, activation='linear'))  
model.fit(X, y, epochs=30, batch\_size=1024)

Получаются следующие результаты:

…Epoch 26/30

162/162 ━━━━━━━━━━━━━━━━━━━━ 6s 37ms/step - loss: 83.2156

Epoch 27/30

162/162 ━━━━━━━━━━━━━━━━━━━━ 6s 34ms/step - loss: 89.1095

Epoch 28/30

162/162 ━━━━━━━━━━━━━━━━━━━━ 6s 37ms/step - loss: 79.4787

Epoch 29/30

162/162 ━━━━━━━━━━━━━━━━━━━━ 5s 32ms/step - loss: 82.5929

Epoch 30/30

162/162 ━━━━━━━━━━━━━━━━━━━━ 5s 30ms/step - loss: 84.9444

5157/5157 ━━━━━━━━━━━━━━━━━━━━ 8s 2ms/step

Mean squared error: 77.5150568800658

RMSE = 8.77

Значения ухудшились не смотря на увеличение количества нейронов

Теперь попробуем добиться лучших результатов благодаря разделению данных на тестовые и тренировочные

Для начала проведём нормализацию данных   
Нормализация данных — это процесс преобразования данных к определенному масштабу, чтобы все признаки имели одинаковый диапазон значений. Это важно для алгоритмов машинного обучения, особенно для нейронных сетей, поскольку различия в масштабах признаков могут отрицательно влиять на обучение модели.

StandardScaler

StandardScaler [19] из библиотеки sklearn [19] используется для стандартизации признаков, удаляя среднее значение и масштабируя их до единичного стандартного отклонения. То есть, после применения StandardScaler [19], каждый признак будет иметь среднее значение 0 и стандартное отклонение 1.

Реализация будет выглядить так   
from sklearn.preprocessing import StandardScaler

StandardScaler scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

Что делает код:

Создание объекта StandardScaler:

scaler = StandardScaler()

Создается объект StandardScaler, который будет использоваться для нормализации данных.

Применение масштабирования к данным:

X\_scaled = scaler.fit\_transform(X)

Метод fit\_transform выполняет два действия:

fit: Вычисляет среднее значение и стандартное отклонение для каждого признака в данных X.

transform: Применяет вычисленные значения, чтобы трансформировать данные, так что каждый признак будет иметь среднее значение 0 и стандартное отклонение 1.

Преимущества нормализации данных:

Ускорение сходимости: Нормализация ускоряет сходимость алгоритмов оптимизации, таких как градиентный спуск, что позволяет модели быстрее достичь оптимального решения.

Стабильность обучения: Нормализация предотвращает доминирование признаков с большими значениями над признаками с меньшими значениями, что делает обучение более стабильным.

Улучшение производительности модели: Некоторые модели машинного обучения (например, нейронные сети, SVM) чувствительны к масштабу данных. Нормализация данных может улучшить их производительность.

Далее мы разделяем данные на обучающую и валидационную выборки

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

Разделение данных:

train\_test\_split — это функция из sklearn.model\_selection, которая используется для разделения данных на обучающие и тестовые (валидационные) выборки.

Аргументы функции:

X\_scaled: Нормализованные данные признаков.

y: Целевая переменная (метка).

test\_size=0.2: Определяет, что 20% данных будут использованы для тестовой (валидационной) выборки, а оставшиеся 80% — для обучающей выборки.

random\_state=42: Фиксирует начальное состояние генератора случайных чисел, что делает разбиение данных воспроизводимым. Используя это значение, вы всегда будете получать одно и то же разбиение данных при каждом запуске кода.

Зачем это нужно:

Нормализация данных:

Улучшает сходимость и производительность модели.

Препятствует доминированию признаков с большими значениями над признаками с меньшими значениями.

Разделение данных:

Обучение модели на одной части данных и тестирование на другой помогает избежать переобучения.

Позволяет объективно оценить, как хорошо модель будет работать на новых, ранее не виденных данных.

Вместе эти шаги помогают построить более стабильную и предсказуемую модель, которая будет лучше работать на реальных данных.

В итоге программа будет выглядить следующим образом:

import pandas as pd  
from keras.models import Sequential  
from keras.layers import Dense, Dropout  
from keras.callbacks import EarlyStopping  
import numpy as np  
from sklearn.preprocessing import StandardScaler  
from sklearn.model\_selection import train\_test\_split  
data = pd.read\_csv("isakgray.csv")  
X = data.iloc[:, 1:]  
y = data.iloc[:, 0]  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  
model = Sequential()  
model.add(Dense(units=1024,activation='relu', input\_shape=(X\_train.shape[1],)))  
model.add(Dense(units=512, activation='relu'))  
model.add(Dense(units=256, activation='relu'))  
model.add(Dense(units=128, activation='relu'))  
model.add(Dense(units=1, activation='linear'))  
model.compile(loss='mean\_squared\_error', optimizer='adam')  
model.fit(X\_train, y\_train, epochs=30, batch\_size=32, validation\_data=(X\_val, y\_val))  
predictions = model.predict(X\_scaled)  
mse = ((predictions[:, 0] - y)\*\*2).mean()  
print("Mean squared error: {}".format(mse))  
print(model.summary())

Результаты

…

Epoch 26/30

4126/4126 ━━━━━━━━━━━━━━━━━━━━ 27s 7ms/step - loss: 75.6894 - val\_loss: 91.3831

Epoch 27/30

4126/4126 ━━━━━━━━━━━━━━━━━━━━ 25s 6ms/step - loss: 73.7841 - val\_loss: 89.8567

Epoch 28/30

4126/4126 ━━━━━━━━━━━━━━━━━━━━ 23s 6ms/step - loss: 73.6789 - val\_loss: 87.4201

Epoch 29/30

4126/4126 ━━━━━━━━━━━━━━━━━━━━ 22s 5ms/step - loss: 72.7564 - val\_loss: 86.0380

Epoch 30/30

4126/4126 ━━━━━━━━━━━━━━━━━━━━ 22s 5ms/step - loss: 72.4331 - val\_loss: 92.0962

5157/5157 ━━━━━━━━━━━━━━━━━━━━ 6s 1ms/step

|  |  |  |
| --- | --- | --- |
| Layer (type) | Output Shape | Param # |
| dense (Dense) | (None, 1024) | 19456 |
| dense\_1 (Dense) | (None, 512) | 524800 |
| dense\_2 (Dense) | (None, 256) | 131328 |
| dense\_3 (Dense) | (None, 128) | 32896 |
| dense\_4 (Dense) | (None, 1) | 129 |

Total params: 2,125,829

Trainable params: 708,609

Non-trainable params: 0

Optimizer params: 1,417,220

RMSE = 8.6

Значения получились лучше чем в прошлый раз, попробуем для изображения большего разрешения например 1920 на 1080, т.е количество обучающих векторов равно 6220800

Пример изображения:



Рисунок 3 Изначальное изображение

Для начала сделаем изображение серым и получем обучающую матрицу для серого изображения



Рисунок 4 полученное серое изображение

Полученная обучающая матрица:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| y | X0 | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 | X9 | X10 | X11 | X12 | X13 | X14 | X15 | X16 | X17 |
| 192 | 189 | 225 | 232 | 237 | 197 | 232 | 177 | 225 | 243 | 208 | 251 | 222 | 143 | 221 | 145 | 223 | 222 | 235 |
| 205 | 190 | 208 | 237 | 235 | 192 | 222 | 225 | 234 | 204 | 180 | 232 | 211 | 189 | 230 | 232 | 222 | 215 | 73 |
| 160 | 185 | 180 | 235 | 192 | 222 | 225 | 243 | 204 | 72 | 37 | 222 | 35 | 190 | 158 | 32 | 235 | 68 | 39 |
| 37 | 37 | 38 | 36 | 160 | 35 | 180 | 72 | 37 | 38 | 211 | 41 | 185 | 32 | 236 | 68 | 39 | 34 | 32 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

Результаты:

Для линейной регрессии получаются mse равно 223.77 RMSE равно 14.96

Для нейронной сети:

Epoch 26/30

806/806 ━━━━━━━━━━━━━━━━━━━━ 18s 22ms/step - loss: 170.0885 - val\_loss: 204.0968

Epoch 27/30

806/806 ━━━━━━━━━━━━━━━━━━━━ 22s 24ms/step - loss: 170.2777 - val\_loss: 208.1074

Epoch 28/30

806/806 ━━━━━━━━━━━━━━━━━━━━ 19s 24ms/step - loss: 166.4235 - val\_loss: 203.7472

Epoch 29/30

806/806 ━━━━━━━━━━━━━━━━━━━━ 15s 18ms/step - loss: 165.9181 - val\_loss: 208.3951

Epoch 30/30

806/806 ━━━━━━━━━━━━━━━━━━━━ 15s 19ms/step - loss: 161.9633 - val\_loss: 208.9354

16120/16120 ━━━━━━━━━━━━━━━━━━━━ 33s 2ms/step

Mean squared error: 166.74729368844802

RMSE = 12.9130

Мы получили результаты намного лучше чем линейная регрессия

Попробуем еще уменьшить mse изменив функции активации:

Добавим в слои функцию активации Leaky ReLU и ELU:

Финальная программа будет выглядить следующим образом:

import pandas as pd  
from keras.models import Sequential  
from keras.layers import Dense, Dropout, LeakyReLU, ELU  
import numpy as np  
from sklearn.preprocessing import StandardScaler  
from sklearn.model\_selection import train\_test\_split  
def lin\_regr(A):  
 y = A[:, 0]  
 x = A[:, 1:]  
 lsq = np.linalg.lstsq(x, y, rcond=None)  
 di = x @ lsq[0] - y  
 su2 = np.sum(di\*di) / len(y)  
 print('mse=%12.8f' % su2)  
 print('w=', lsq[0])  
data = pd.read\_csv("TestGray.csv")  
X = data.iloc[:, 1:]  
y = data.iloc[:, 0]  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  
lin\_regr(data.values)  
model = Sequential()  
model.add(Dense(units=2048, activation=LeakyReLU()))  
model.add(Dense(units=1024, activation=ELU()))  
model.add(Dense(units=512, activation='relu'))  
model.add(Dense(units=256, activation='relu'))  
model.add(Dense(units=128, activation='relu'))  
model.add(Dense(units=1, activation='linear'))  
model.compile(loss='mean\_squared\_error', optimizer='adam')  
model.fit(X\_train, y\_train, epochs=30, batch\_size=512, validation\_data=(X\_val, y\_val))  
predictions = model.predict(X\_scaled)  
mse = ((predictions[:, 0] - y)\*\*2).mean()  
print("Mean squared error: {}".format(mse))  
print(model.summary())

Результаты:

Для линейной регрессии получаются mse равно 223.77 RMSE равно 14.96

Для нейронной сети:

...

poch 26/30

806/806 ━━━━━━━━━━━━━━━━━━━━ 55s 68ms/step - loss: 166.6045 - val\_loss: 205.8954

Epoch 27/30

806/806 ━━━━━━━━━━━━━━━━━━━━ 58s 72ms/step - loss: 163.4386 - val\_loss: 208.7073

Epoch 28/30

806/806 ━━━━━━━━━━━━━━━━━━━━ 49s 61ms/step - loss: 159.5636 - val\_loss: 207.2200

Epoch 29/30

806/806 ━━━━━━━━━━━━━━━━━━━━ 49s 60ms/step - loss: 156.1343 - val\_loss: 212.0849

Epoch 30/30

806/806 ━━━━━━━━━━━━━━━━━━━━ 48s 60ms/step - loss: 152.2492 - val\_loss: 208.5550

16120/16120 ━━━━━━━━━━━━━━━━━━━━ 41s 3ms/step

Mean squared error: 158.7427883767382

|  |  |  |
| --- | --- | --- |
| Layer (type) | Output Shape | Param # |
| dense (Dense) | (None, 2048) | 38912 |
| dense\_1 (Dense) | (None, 1024) | 2098176 |
| dense\_2 (Dense) | (None, 512) | 524800 |
| dense\_3 (Dense) | (None, 256) | 131328 |
| dense\_4 (Dense) | (None, 128) | 32896 |
| dense\_5 (Dense) | (None, 1) | 129 |

Total params: 8,478,725

Trainable params: 2,826,241

Non-trainable params: 0

Optimizer params: 5,652,484

RMSE = 12.599

Можно подвести вывод, что нейронная сеть справляется лучше чем обычная линейная регрессия.

Подведём вывод для задачи регрессии методом прогнозирования яркости точки на основе предыдущих

В ходе решения задачи прогнозирования значений с использованием машинного обучения на языке Python был проведён ряд экспериментов с различными моделями и параметрами с целью улучшения точности прогнозов.

Исходная задача включала в обучающую матрицу, где каждая строка представляла собой значения признаков (x0, x1, ..., x16) и соответствующее прогнозируемое значение (y). Основной целью было построение модели, способной предсказывать y на основе входных признаков.

Изначально использовались линейная регрессия и нейронные сети с несколькими слоями. Первоначальные результаты показали, что RMSE для линейной регрессии составляло 9.8, а для нейронной сети - 8.83. Далее были предприняты попытки улучшить точность нейронной сети путем изменения архитектуры, количества нейронов и параметров обучения. Однако, большие изменения в архитектуре и размере пакета (batch\_size) не привели к существенному улучшению результатов.

Дополнительно был проведён эксперимент с разделением данных на тренировочный и тестовый наборы, что также не привело к значительному улучшению модели.

Стандартизация данных и увеличение количества нейронов также не привели к улучшению RMSE, и в некоторых случаях даже привели к увеличению времени обучения.

Затем была проведена попытка стандартизации и изменения функций активации, что привело к небольшому улучшению RMSE.

В итоге, нейронная сеть показала более высокую точность по сравнению с линейной регрессией, и использование оптимальных параметров и архитектуры может значительно повысить эффективность модели. Однако, важно подчеркнуть, что выбор оптимальной модели зависит от специфики данных и требует тщательного тестирования и анализа результатов.

# Сжатие изображения с помощью нейронных сетей методом прогнозирования яркости точки на основе предыдущих

Сжатие изображений с использованием нейронных сетей – это активно исследуемая область в компьютерном зрении и обработке изображений. Нейронные сети могут использоваться для различных задач сжатия, таких как сжатие с потерями и без потерь.

Сжатие с потерями:

1.Нейронные сети могут быть обучены для кодирования изображений с потерями, где потери качества минимизируются с использованием различных техник, таких как вариационные автокодировщики или глубокие сверточные сети.

2.Результаты сжатия с потерями могут быть весьма эффективными, обеспечивая значительное сжатие с минимальной потерей качества изображения, что особенно важно для передачи данных через сеть или хранения на устройствах с ограниченным пространством.

Сжатие без потерь:

1.Нейронные сети также могут применяться для сжатия без потерь. Это может включать в себя методы, основанные на обучении представлений, которые могут изучать эффективные способы кодирования и декодирования изображений без потерь.

2.Результаты сжатия без потерь могут быть менее эффективными в терминах степени сжатия по сравнению со сжатием с потерями, но они сохраняют качество изображения полностью.

Общие результаты сжатия изображений с помощью нейронных сетей варьируются в зависимости от конкретной архитектуры сети, используемых методов обучения и требуемых характеристик сжатия (например, степень сжатия, сохранение качества и скорость обработки). В целом, нейронные сети показывают потенциал для создания более эффективных и интеллектуальных методов сжатия изображений, чем традиционные алгоритмы.

Каких результатов можно добится благодаря сжатию нейронными сетями:

1.Улучшенное качество сжатия: Нейронные сети позволяют создавать более эффективные алгоритмы сжатия, сохраняя при этом высокое качество изображения. Это достигается за счет использования глубоких сверточных сетей для изучения особенностей изображений и разработки более интеллектуальных методов сжатия.

2.Сверхразрешение: Нейронные сети могут использоваться для увеличения разрешения изображений после сжатия. Это позволяет сохранить детали и качество изображения при сжатии.

3.Адаптивное сжатие: Нейронные сети позволяют создавать алгоритмы, которые могут адаптироваться к различным типам изображений и контенту, обеспечивая более эффективное сжатие для конкретных сценариев использования.

4.Прогрессивное сжатие: Некоторые исследования сосредоточены на разработке методов прогрессивного сжатия с использованием нейронных сетей. Это позволяет постепенно улучшать качество изображения по мере его загрузки, что особенно полезно для онлайн-просмотра изображений.

5.Малоразмерные модели для мобильных устройств: Создание небольших и эффективных моделей нейронных сетей для сжатия изображений является активной областью исследований, поскольку это позволяет применять такие методы на мобильных устройствах с ограниченными вычислительными ресурсами.

В данной реализации мы будем сжимать картинку без потери качества.

Алгоритм выглядит следующим образом:

1.Подготовка изображеия. На вход получаем изображение(цветное).

2.Преобразование изображения в серое

3.Создание обучающей матрицы.Для серого изображения строим обучающую матрицу.

4.Подготовка данных

5.Обучение модели.На обучающей матрицы делаем предсказания.

6.Сохраняем матрицу предсказаний в формате целых чисел(np.uint8).

7.Вычисляем матрицу отклонений, т.е из обучающей матрицы вычитаем округлённую матрицу.

8.Сохраняем полученную матрицу в .npy(не сжатый пайтоном) и .npz(сжатый пайтоном соответственно) форматах.

9.Анализ данных.Для проверки сделаем 2 функции: 1 будет считать распределение т.е сколько положительных, отрицательных и нулевых значений. 2 функция строит гистограмму распределения.

10.Далее не сжатый .npy файл сожмём 7-zip с максимально возможным сжатием.

11.Подведем итоги насколько насколько лучше или хуже получился размер сжатого .npy файла по сравнению с исходным png.

Реализация алгоритма на python:

Первым шагом в реализации алгоритма является подготовка изображения. Этот шаг включает в себя получение цветного изображения на входе и его преобразование в серое изображение. Это важно, так как дальнейшая обработка и анализ будут выполнены на сером изображении. В этом разделе мы будем использовать библиотеки numpy и PIL для работы с изображениями.

Шаги:

1.Импорт необходимых библиотек.

2.Загрузка цветного изображения.

3.Преобразование цветного изображения в серое.

Шаг 1: Импорт необходимых библиотек.

Для начала нам нужно импортировать библиотеки, которые мы будем использовать для загрузки и преобразования изображения.

import numpy as np

from PIL import Image

Шаг 2: Загрузка цветного изображения

Мы будем использовать библиотеку PIL (Python Imaging Library) для загрузки изображения. PIL позволяет легко работать с изображениями, включая их загрузку, преобразование и сохранение.

def load\_image(image\_path):

# Загрузка изображения с указанного пути

img\_data = Image.open(image\_path)

# Преобразование изображения в массив numpy

img\_arr = np.array(img\_data)

return img\_arr

В этой функции load\_image, мы используем Image.open для открытия изображения и np.array, чтобы преобразовать изображение в массив numpy.

Шаг 3: Преобразование цветного изображения в серое

Для преобразования цветного изображения в серое, мы будем использовать стандартную формулу для расчёта яркости, которая учитывает вклады каждого цветового канала (красного, зелёного и синего). Эта формула выглядит следующим образом:



def convert\_to\_gray(img\_arr):

# Используем формулу для преобразования в серый цвет

gray\_img = np.round(0.299 \* img\_arr[:, :, 0] + 0.587 \* img\_arr[:, :, 1] + 0.114 \* img\_arr[:, :, 2])

# Преобразуем результат в массив целых чисел

gray\_img = np.array(gray\_img, dtype=int)

return gray\_img

В этой функции convert\_to\_gray, мы используем формулу для преобразования цветного изображения в серое, а затем преобразуем результат в массив целых чисел для дальнейшей обработки.

Полный код для подготовки изображения

def prepare\_image(image\_path):

# Шаг 1: Загрузка изображения

img\_arr = load\_image(image\_path)

# Шаг 2: Преобразование изображения в серое

gray\_img = convert\_to\_gray(img\_arr)

return gray\_img

# Пример использования функции

image\_path = '1024px-Sunflower\_from\_Silesia2.png'

gray\_image = prepare\_image(image\_path)

Создание обучающей матрицы

На этом этапе мы будем строить обучающую матрицу на основе серого изображения. Обучающая матрица будет использоваться для обучения модели, которая будет предсказывать значения точек. Мы будем использовать значения точек и их окрестности для создания признаков для обучения как и в задаче регрессии.

Шаги:

1.Импорт необходимых библиотек.

2.Создание обучающей матрицы на основе серого изображения.

Шаг 1: Импорт необходимых библиотек

Мы уже импортировали numpy, поэтому дополнительно ничего импортировать не нужно.

Шаг 2: Создание обучающей матрицы

Создадим функцию tteach18, которая будет строить обучающую матрицу. Эта функция будет использовать значение текущей точки и её окрестностей для создания обучающих данных.

def tteach18(a):

step\_y = 1

step\_x = 1

my, mx = a.shape

b = np.zeros(((my - 3) \* (mx - 6), 19), dtype=np.uint8)

i = 0

for y in range(3, my, step\_y):

for x in range(3, mx - 3, step\_x):

b[i, 0] = a[y, x]

b[i, 1] = a[y, x - 1]

b[i, 2] = a[y - 1, x]

b[i, 3] = a[y - 1, x - 1]

b[i, 4] = a[y - 1, x + 1]

b[i, 5] = a[y, x - 2]

b[i, 6] = a[y - 2, x]

b[i, 7] = a[y - 1, x - 2]

b[i, 8] = a[y - 2, x - 1]

b[i, 9] = a[y - 2, x + 1]

b[i, 10] = a[y - 1, x + 2]

b[i, 11] = a[y - 2, x - 2]

b[i, 12] = a[y - 2, x + 2]

b[i, 13] = a[y, x - 3]

b[i, 14] = a[y - 3, x]

b[i, 15] = a[y - 1, x - 3]

b[i, 16] = a[y - 3, x - 1]

b[i, 17] = a[y - 3, x + 1]

b[i, 18] = a[y - 1, x + 3]

i += 1

return b[:i]

Функция tteach18 принимает серое изображение в качестве входных данных и возвращает обучающую матрицу. В этой матрице каждый ряд представляет собой набор признаков, состоящий из значения текущей точки и её окрестностей.

Подготовка данных

На этом этапе мы подготовим данные для обучения модели. Мы будем использовать подготовленное серое изображение для создания обучающей матрицы.

Создадим функцию prepare\_data, которая будет использовать функции load\_image, convert\_to\_gray и tteach18 для подготовки данных.

def prepare\_data(image\_path):

# Шаг 1: Загрузка изображения и преобразование в серое

img\_data = Image.open(image\_path)

img\_arr = np.array(img\_data)

gray\_img = convert\_to\_gray(img\_arr)

# Шаг 2: Создание обучающей матрицы

train\_matrix = tteach18(gray\_img)

return train\_matrix

# Пример использования функции

image\_path = '1024px-Sunflower\_from\_Silesia2.png'

train\_matrix = prepare\_data(image\_path)

Функция prepare\_data принимает путь к изображению, загружает его, преобразует в серое изображение и создаёт обучающую матрицу, используя описанные выше шаги. В итоге, мы получаем матрицу, готовую для обучения модели.

Обучение модели

На этом этапе мы будем использовать обучающую матрицу для обучения модели. Мы будем использовать нейронную сеть, созданную с помощью tensorflow.keras(созданную и протестированную для решения задачи регрессии), чтобы обучить модель предсказывать значения точки на основе её окружения.

Шаги:

1.Импорт необходимых библиотек.

2.Определение и компиляция модели.

3.Обучение модели.

Шаг 1: Импорт необходимых библиотек

Мы уже импортировали numpy и PIL. Теперь нам нужно импортировать библиотеки tensorflow.keras для создания и обучения модели.

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, ELU, LeakyReLU

Шаг 2: Определение и компиляция модели

Создадим функцию train\_model, которая определяет архитектуру модели и компилирует её. Мы будем использовать полносвязные слои и различные функции активации для создания модели.

def train\_model(train\_matrix):

#Разделение обучающей матрицы на признаки (X) и целевую переменную(y)

X\_train = train\_matrix[:, 1:] y\_train = train\_matrix[:, 0]

# Определение модели

model=Sequential()

model.add(Dense(units=2048,activation=LeakyReLU())) model.add(Dense(units=512,activation=ELU())) model.add(Dense(units=256,activation='relu')) model.add(Dense(units=128,activation='relu')) model.add(Dense(units=64,activation='relu'))

model.add(Dense(units=32,activation='relu'))

model.add(Dense(units=16, activation='relu'))

model.add(Dense(units=1, activation='linear'))

# Компиляция модели

model.compile(loss='mean\_squared\_error', optimizer='adam')

# Обучение модели

model.fit(X\_train, y\_train, epochs=15, batch\_size=4096)

return model

# Пример использования функции

train\_matrix = prepare\_data(image\_path)

model = train\_model(train\_matrix)

Обучение модели

Функция train\_model принимает обучающую матрицу, разделяет её на признаки и целевую переменную, определяет и компилирует модель, а затем обучает её. После обучения функция возвращает обученную модель.

Сохранение матрицы предсказаний

На этом этапе мы будем использовать обученную модель для предсказания значений пикселей и сохранения матрицы предсказаний в формате целых чисел (np.uint8).

Шаги:

1.Использование обученной модели для предсказания значений.

2.Сохранение матрицы предсказаний.

Шаг 1: Использование обученной модели для предсказания значений

Создадим функцию calculate\_deviation\_matrix, которая использует обученную модель для предсказания значений пикселей на основе серого изображения и вычисляет матрицу отклонений.

def calculate\_deviation\_matrix(image\_path, model):

img\_data = Image.open(image\_path)

img\_arr = np.array(img\_data)

gray\_img = convert\_to\_gray(img\_arr)

height, width = gray\_img.shape

deviations = np.zeros((height, width), dtype=int)

samples = []

# Подготовка выборок для предсказания

for y in range(3, height):

for x in range(3, width - 3):

sample = [

gray\_img[y, x], gray\_img[y, x - 1], gray\_img[y - 1, x],

gray\_img[y - 1, x - 1], gray\_img[y - 1, x + 1], gray\_img[y, x - 2],

gray\_img[y - 2, x], gray\_img[y - 1, x - 2], gray\_img[y - 2, x - 1],

gray\_img[y - 2, x + 1], gray\_img[y - 1, x + 2], gray\_img[y - 2, x - 2],

gray\_img[y - 2, x + 2], gray\_img[y, x - 3], gray\_img[y - 3, x],

gray\_img[y - 1, x - 3], gray\_img[y - 3, x - 1], gray\_img[y - 3, x + 1],

gray\_img[y - 1, x + 3]

]

samples.append(sample[1:])

samples\_np = np.array(samples)

predictions = model.predict(samples\_np)

rounded\_predictions = np.round(predictions).astype(int).flatten()

# Вычисление матрицы отклонений

i = 0

for y in range(3, height):

for x in range(3, width - 3):

deviation = int(gray\_img[y, x]) - rounded\_predictions[i]

deviations[y, x] = deviation

i += 1

for y in range(3):

for x in range(width):

if x == 0:

deviations[y, x] = gray\_img[y, x]

else:

deviations[y, x] = gray\_img[y, x] - gray\_img[y, x - 1]

for y in range(3, height):

for x in range(3):

if y == 0:

deviations[y, x] = gray\_img[y, x]

else:

deviations[y, x] = gray\_img[y, x] - gray\_img[y - 1, x]

deviations[0, 0] = gray\_img[0, 0]

return deviations

# Пример использования функции

deviation\_matrix = calculate\_deviation\_matrix(image\_path, model)

Основные шаги функции

Загрузка изображения и преобразование в серое

Функция загружает цветное изображение с указанного пути и преобразует его в массив.

Изображение преобразуется в градации серого.

Инициализация матрицы отклонений

Создаётся пустая матрица отклонений того же размера, что и серое изображение.

Подготовка выборок для предсказания

Проходит по каждй точик изображения (начиная с четвертого ряда и столбца) и собирает окрестности точки в выборки.

Для каждой точки формируется массив из значений его соседей. Эти массивы добавляются в список samples.

Предсказание значений точки

Преобразует список выборок в массив samples\_np.

Использует обученную модель для предсказания значений точки на основе этих выборок.

Округляет предсказанные значения и преобразует их в целые числа.

Вычисление матрицы отклонений

Проходит по каждой точки, для которого делалось предсказание, и вычисляет разницу между фактическим значением точки и предсказанным значением.

Заполняет матрицу отклонений этими значениями разности.

Для первых трёх строк и столбцов матрица отклонений заполняется разностями значений точки с их соседями.

Сохранение матрицы предсказаний

Создадим функцию save\_matrix, которая будет сохранять матрицу предсказаний в формате целых чисел (np.uint8).

def save\_matrix(matrix, filename, compressed=False):

if compressed:

np.savez\_compressed(filename, matrix)

else:

np.save(filename, matrix)

# Пример использования функции

save\_matrix(deviation\_matrix, 'matrix3', compressed=False)

save\_matrix(deviation\_matrix, 'matrix3\_compressed', compressed=True)

Функция save\_matrix принимает матрицу и сохраняет её в файл. Если параметр compressed установлен в True, матрица сохраняется в сжатом формате .npz. В противном случае она сохраняется в несжатом формате .npy.

Анализ данных

Функция для подсчёта распределения значений.

Эта функция будет подсчитывать количество положительных, отрицательных и нулевых значений в матрице отклонений.

def count\_values(matrix):

positive\_count = np.sum(matrix > 0)

negative\_count = np.sum(matrix < 0)

zero\_count = np.sum(matrix == 0)

return positive\_count, negative\_count, zero\_count

Пример использования:

positive\_count,negative\_count,zero\_count=count\_values(deviation\_matrix)

print(f"Количество положительных значений: {positive\_count}")

print(f"Количество отрицательных значений: {negative\_count}")

print(f"Количество нулевых значений: {zero\_count}")

Функция для построения гистограммы распределения

Эта функция будет строить гистограмму распределения значений в матрице отклонений.

import matplotlib.pyplot as plt

def plot\_histogram(matrix, value\_range=(-10, 10)):

values = matrix.flatten()

values = values[(values >= value\_range[0]) & (values <= value\_range[1])]

plt.hist(values, bins=range(value\_range[0], value\_range[1] + 1), edgecolor='black', align='left')

plt.title('Гистограмма отклонений значений')

plt.xlabel('Значения')

plt.ylabel('Количество')

plt.grid(True)

plt.show()

Пример использования:

plot\_histogram(deviation\_matrix, value\_range=(-10, 10))

Сжатие .npy файла с помощью 7-Zip

Для этого шага мы используем библиотеку py7zr для сжатия файла.

import py7zr

def compress\_file(input\_file, output\_archive):

with py7zr.SevenZipFile(output\_archive, 'w') as archive:

archive.write(input\_file, arcname=input\_file)

# Пример использования

input\_file = 'matrix3.npy'

output\_archive = 'compressed\_file.7z'

compress\_file(input\_file, output\_archive)

Итоговая программа выглядит следуюшщим образом:

import numpy as np  
from PIL import Image  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense, ELU, LeakyReLU  
import numpy as np  
import matplotlib.pyplot as plt  
import py7zr  
def plot\_histogram(matrix, value\_range=(-10, 10)):  
 values = matrix.flatten()  
 values = values[(values >= value\_range[0]) & (values <= value\_range[1])]  
 plt.hist(values, bins=range(value\_range[0], value\_range[1] + 1), edgecolor='black',align='left')  
 plt.title('Гистограмма отклонений значений')  
 plt.xlabel('Значения')  
 plt.ylabel('кол-во')  
 plt.grid(True)  
 plt.show()  
def a\_gray(a):  
 y = np.round(0.299 \* a[:, :, 0] + 0.587 \* a[:, :, 1] + 0.114 \* a[:, :, 2])  
 return np.array(y, dtype=int)  
def tteach18(a):  
 step\_y = 1  
 step\_x = 1  
 my, mx = a.shape  
 b = np.zeros(((my - 3) \* (mx - 6), 19), dtype=np.uint8)  
 i = 0  
 for y in range(3, my, step\_y):  
 for x in range(3, mx - 3, step\_x):  
 b[i, 0] = a[y, x]  
 b[i, 1] = a[y, x - 1]  
 b[i, 2] = a[y - 1, x]  
 b[i, 3] = a[y - 1, x - 1]  
 b[i, 4] = a[y - 1, x + 1]  
 b[i, 5] = a[y, x - 2]  
 b[i, 6] = a[y - 2, x]  
 b[i, 7] = a[y - 1, x - 2]  
 b[i, 8] = a[y - 2, x - 1]  
 b[i, 9] = a[y - 2, x + 1]  
 b[i, 10] = a[y - 1, x + 2]  
 b[i, 11] = a[y - 2, x - 2]  
 b[i, 12] = a[y - 2, x + 2]  
 b[i, 13] = a[y, x - 3]  
 b[i, 14] = a[y - 3, x]  
 b[i, 15] = a[y - 1, x - 3]  
 b[i, 16] = a[y - 3, x - 1]  
 b[i, 17] = a[y - 3, x + 1]  
 b[i, 18] = a[y - 1, x + 3]  
 i += 1  
 return b[:i]  
def prepare\_data(image\_path):  
 img\_data = Image.open(image\_path)  
 img\_arr = np.array(img\_data)  
 gray\_img = a\_gray(img\_arr)  
 train\_matrix = tteach18(gray\_img)  
 return train\_matrix  
def train\_model(train\_matrix):  
 X\_train = train\_matrix[:, 1:]  
 y\_train = train\_matrix[:, 0]  
 model = Sequential()  
 model.add(Dense(units=2048, activation=LeakyReLU()))  
 model.add(Dense(units=512, activation=ELU()))  
 model.add(Dense(units=256, activation='relu'))  
 model.add(Dense(units=128, activation='relu'))  
 model.add(Dense(units=64, activation='relu'))  
 model.add(Dense(units=32, activation='relu'))  
 model.add(Dense(units=16, activation='relu'))  
 model.add(Dense(units=1, activation='linear'))  
 model.compile(loss='mean\_squared\_error', optimizer='adam')  
 model.fit(X\_train, y\_train, epochs=15, batch\_size=4096)  
 return model  
def calculate\_deviation\_matrix(image\_path, model):  
 img\_data = Image.open(image\_path)  
 img\_arr = np.array(img\_data)  
 gray\_img = a\_gray(img\_arr)  
 height, width = gray\_img.shape  
 deviations = np.zeros((height, width), dtype=int)  
 samples = []  
 for y in range(3, height):  
 for x in range(3, width - 3):  
 sample = [  
 gray\_img[y, x], gray\_img[y, x - 1], gray\_img[y - 1, x],  
 gray\_img[y - 1, x - 1], gray\_img[y - 1, x + 1], gray\_img[y, x - 2],  
 gray\_img[y - 2, x], gray\_img[y - 1, x - 2], gray\_img[y - 2, x - 1],  
 gray\_img[y - 2, x + 1], gray\_img[y - 1, x + 2], gray\_img[y - 2, x - 2],  
 gray\_img[y - 2, x + 2], gray\_img[y, x - 3], gray\_img[y - 3, x],  
 gray\_img[y - 1, x - 3], gray\_img[y - 3, x - 1], gray\_img[y - 3, x + 1],  
 gray\_img[y - 1, x + 3]  
 ]  
 samples.append(sample[1:])  
 samples\_np = np.array(samples)  
 predictions = model.predict(samples\_np)  
 rounded\_predictions = np.round(predictions).astype(int).flatten()  
 i = 0  
 for y in range(3, height):  
 for x in range(3, width - 3):  
 deviation = int(gray\_img[y, x]) - rounded\_predictions[i]  
 deviations[y, x] = deviation  
 i += 1  
 for y in range(3):  
 for x in range(width):  
 if x == 0:  
 deviations[y, x] = gray\_img[y, x]  
 else:  
 deviations[y, x] = gray\_img[y, x] - gray\_img[y, x - 1]  
 for y in range(3, height):  
 for x in range(3):  
 if y == 0:  
 deviations[y, x] = gray\_img[y, x]  
 else:  
 deviations[y, x] = gray\_img[y, x] - gray\_img[y - 1, x]  
 deviations[0, 0] = gray\_img[0, 0]  
 return deviations  
def save\_matrix\_to\_csv(matrix, filename):  
 np.savetxt(filename, matrix, delimiter=',', fmt='%d')  
def count\_values(matrix):  
 positive\_count = np.sum(matrix > 0)  
 negative\_count = np.sum(matrix < 0)  
 zero\_count = np.sum(matrix == 0)  
 return positive\_count, negative\_count, zero\_count  
def save\_matrix(matrix, filename, compressed=False):  
 if compressed:  
 np.savez\_compressed(filename, matrix)  
 else:  
 np.save(filename, matrix)  
image\_path = '1024px-Sunflower\_from\_Silesia2.png'  
train\_matrix = prepare\_data(image\_path)  
model = train\_model(train\_matrix)  
deviation\_matrix = calculate\_deviation\_matrix(image\_path, model)  
save\_matrix\_to\_csv(deviation\_matrix, 'matrix3.csv')  
save\_matrix(deviation\_matrix, 'matrix3', compressed=True)  
save\_matrix(deviation\_matrix, 'matrix3', compressed=False)  
positive\_count, negative\_count, zero\_count = count\_values(deviation\_matrix)  
print(f"Количество положительных значений: {positive\_count}")  
print(f"Количество отрицательных значений: {negative\_count}")  
print(f"Количество нулевых значений: {zero\_count}")  
plot\_histogram(deviation\_matrix, value\_range=(-10, 10))  
input\_file = 'matrix3.npy'  
output\_archive = 'compressed\_file.7z'  
with py7zr.SevenZipFile(output\_archive, 'w') as archive:  
 archive.write(input\_file, arcname='matrix3.npy.npy')

Этот код выполняет все этапы обработки изображения, обучения модели, вычисления и анализа матрицы отклонений, а также сжатия файла и сравнения размеров до и после сжатия.

Пример работы программы:

Для изображения разрешения 1024 на 714 серого цвета



Рисунок 5 изображение для сжатия

Имеющего размер 313Кб (320887 байт)

Матрица отклонений выглядит следующим образом:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 99 | 1 | 1 | 1 | 0 | -1 | -1 | -1 | -1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 98 | 1 | 1 | 0 | 0 | 0 | -1 | -1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 98 | 0 | 1 | 0 | 0 | 0 | -1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

Полученный npy файл имее размер 2.78 МБ (2 924 672 байт)

Полученный с помощью 7-zip сжатия файл имеет размер  292373  байт.

Npz файл имеет размер 505 КБ (517 472 байт)

Количество положительных значений: 218523  
Количество отрицательных значений: 230962  
Количество нулевых значений: 281651

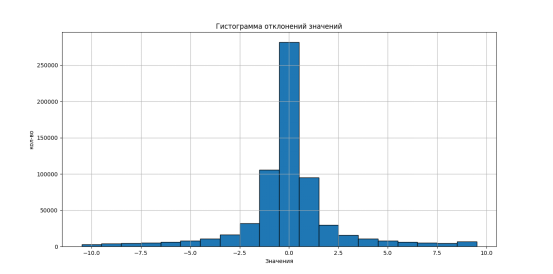


Рисунок 6 Гистограмма отклонений значений(1)

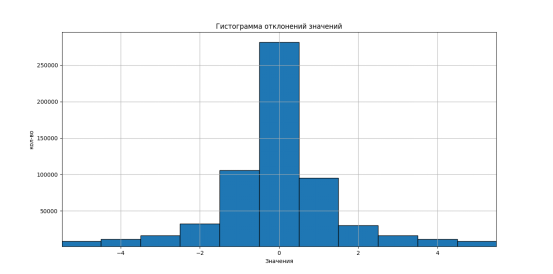


Рисунок 7 Гистограмма отклонений значений(2)

Рассмотрим результаты, полученные при сжатии изображения:

Размер изначального серого изображения: 313 КБ (320,887 байт)

Размер полученного npy файла: 2.78 МБ (2,924,672 байт)

Размер сжатого файла с помощью 7-zip: 292,373 байт

Размер npz файла: 505 КБ (517,472 байт)

Анализ результатов

1. Размер изначального серого изображения

Изначальное серое изображение имеет размер 313 КБ. Этот размер является базовой точкой для сравнения с размерами других файлов.

2. Размер полученного npy файла

Полученный npy файл, который содержит матрицу отклонений, имеет размер 2.78 МБ. Это значительно больше по сравнению с исходным изображением.

Причина увеличения размера заключается в том, что npy файл хранит данные в несжатом виде, что может занимать много места, особенно если данные содержат много числовых значений.

3. Размер сжатого файла с помощью 7-zip

После сжатия npy файла с помощью 7-zip, размер файла уменьшился до 292,373 байт, что чуть меньше исходного размера изображения.

Это показывает, что 7-zip эффективно сжимает данные, уменьшая размер почти на порядок по сравнению с несжатым npy файлом.

4.Размер npz файла:

npz файл, который использует сжатие при сохранении, имеет размер 505 КБ. Это больше, чем исходное изображение, но значительно меньше, чем несжатый npy файл.

Заключение

Эффективность сжатия: Метод сжатия с помощью 7-zip показал наилучшие результаты, уменьшив размер файла до уровня чуть ниже исходного изображения. Это подтверждает, что 7-zip является эффективным инструментом для сжатия таких данных.

Сравнение методов сжатия: npz файл, хотя и применяет сжатие, все же больше по размеру по сравнению с файлом, сжатым с помощью 7-zip. Это может быть связано с различиями в алгоритмах сжатия, используемых в npz и 7-zip.

Практическая значимость: Использование методов сжатия, таких как 7-zip, может значительно сократить объем хранимых данных, что особенно важно при работе с большими наборами данных.

Эти результаты подчеркивают важность выбора подходящего метода сжатия для различных типов данных и могут быть полезны для дальнейших исследований и оптимизации алгоритмов сжатия.

# Заключение

Метод прогнозирования яркости точки на основе предыдущих значений отлично справился как с задачей регрессии, так и с задачей сжатия изображений.

Для решение задачи регрессии:

Подход прогнозирования яркости точки через значения предыдущих точек позволяет учесть сложные зависимости в данных. Использование нейронных сетей для этой задачи привело к лучшим результатам, чем линейная регрессия. Нейронные сети, в отличие от линейных моделей, способны улавливать нелинейные закономерности в данных благодаря нелинейным активациям и глубоким слоям. Это позволяет модели эффективно адаптироваться к сложным взаимосвязям между точками изображения и предсказывать яркость точек с высокой точностью. Точные предсказания важны для широкого спектра приложений, включая обработку изображений, компьютерное зрение и медицинскую диагностику.

Сжатие изображений:

Применение метода прогнозирования яркости точек также позволяет эффективно сжимать изображения. Создание матрицы отклонений, основанной на предсказанных значениях яркости, позволяет сохранить основную информацию об изображении с минимальным количеством данных. Это позволяет сократить размер файла изображения, что является критически важным в современных приложениях, требующих передачи и хранения больших объёмов данных. Применение сжатия с использованием 7-zip дополнительно уменьшает размер файла, делая его более компактным и экономичным для хранения и передачи.

Можно сделать вывод, что метод прогнозирования яркости точек представляет собой мощный инструмент как для решения задач регрессии, так и для сжатия изображений. Нейронные сети, используемые в этом методе, обеспечивают высокую точность предсказаний и эффективность сжатия, что делает его привлекательным выбором для широкого спектра задач обработки изображений. Эти результаты подтверждают потенциал метода прогнозирования яркости точек и его значимость для различных областей применения, от компьютерного зрения до сжатия данных.

# Список литературы

1. Капаца Е. Машинное обучение доступным языком / Е. Капаца — «Автор», 2023
2. Мюллер А. Введение в машинное обучение с помощью Python : руководство для специалистов по работе с данными / Андреас Мюллер, Сара Гвидо. - Москва [и др.] : Диалектика, 2019. - 472.
3. Создаем нейронную сеть. : Пер. с англ. — СПб. : ООО “Альфа-книга”, 2017. — 272 с. : ил. — Парал. тит. англ.
4. Нейронные сети: полный курс, 2-е издание. : Пер. с англ. “Вильямс”, 2(Ш. — 1104 с. : ил. — Парад. тит. англ. М. : Издательский дом
5. Артем Демиденко Машинное обучение. Погружение в технологию, 2023
6. Гольдберг Й. Нейросетевые методы в обработке естественного языка / пер. с анг. А. А. Слинкина. — М.: ДМК Пресс, 2019. — 282 с.: ил.
7. Гласснер Э. Глубокое обучение без математики. Т. 1: Основы / пер. с анг. В. А. Яроцко- го. – М.: ДМК Пресс, 2019. – 584 с.: ил.
8. Гласснер Э. Глубокое обучение без математики.Т. 2: Практика / пер. с анг.В. А. Яроцкого. – М.: ДМК Пресс, 2020. – 610 с.: ил.
9. Ростовцев В.С. Искусственные нейронные сети: учебник / В.С. Ростовцев. – Киров: Изд-во ВятГУ, 2014. – 208 с. Э4743
10. Уатт, Дж. Машинное обучение: основы, алгоритмы и практика применения:Пер. с англ. / Дж. Уатт, Р. Борхани, А. Катсаггелос. — СПб.: БХВ-Петербург, 2022.—640 с.: ил.
11. Микелуччи У. Прикладное глубокое обучение. Подход к пониманию глубоких нейронных сетей на основе метода кейсов: Пер. с англ. — СПб.: БХВ-Петербург, 2020. — 368 с.: ил.
12. Долганов, Антон Юрьевич. Базовые алгоритмы машинного обучения на языке Python : учебно-ме-тодическое пособие / А.Ю. Долганов, М.В. Ронкин, А.В. Созыкин ; М-во науки и высшего образования РФ.— Екатеринбург : Изд-во Урал. ун-та, 2023.— 124 с.
13. Шакла Н. Машинное обучение и TeпsorFlow. - СПб.: Питер, 2019. - 336 с.: ил. - (Серия «Библиотека программиста»).
14. Мюллер А., Гидо С.Введение в машинное обучение с помощью Python: Вильямс, 2017.
15. Лекун Я. Как учится машина: Революция в области нейронных сетей и глубокого обучения / Ян Лекун. — Пер. с фр. — М.: Альпина ПРО, 2021.
16. Дайзенрот Марк Питер, Альдо Фейзал А., Чен Сунь Он. Математика в машинном обучении. — СПб.: Питер, 2024. — 512 с.: ил. — (Се-рия «Для профессионалов»).
17. Элбон Крис. Машинное обучение с использованием Python. Сборник рецептов: Пер. с англ. — СПб.: БХВ-Петербург, 2019. — 384 с.: ил.
18. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. — СПб.: Питер, 2018. — 480 с.: ил. — (Серия «Библиотека программиста»).
19. TensorFlow. Материал из документации – TensorFlow для Python. URL: https://www.tensorflow.org/learn?hl=ru (дата обращения: 10.03.2024).
20. TensorFlow. Руководство по Keras [Электронный ресурс]. URL: <https://www.tensorflow.org/guide/keras?hl=ru> (дата обращения: 07.06.2024).